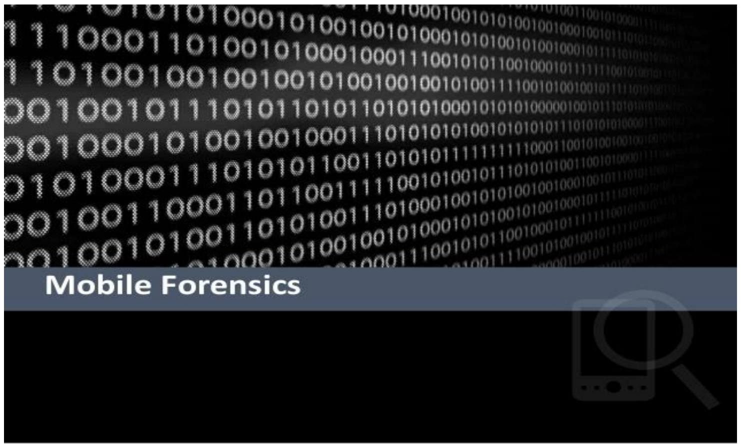
UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL SYSTEMS

POSTGRADUATE PROGRAMME

Economic Management and Digital Systems Security



Smartphone Forensics & Data Acquisition

DISSERTATION

Pachigiannis Panagiotis
MTE1219

2015

Contents

Devoted,

To my parents and my whole family because throughout my course, the easy and the difficult, everything I had on my side.

My support and unconditional love for me were the inspiration for what I've done so far and I thank them from my heart!!

Acknowledgement

With the completion of this thesis, I would like to thank my supervisor **Konstantinos Lambrinoudakis**[1] **for** the award of the matter in confidence he showed me, the knowledge offered me during the preparation of thesis and for extremely useful and constructive suggestions during the writing of.

Furthermore, I would like to thank all the postgraduate's teachers of **"Economic Management & Digital Security Systems"** of direction "Digital Systems Security" of the Department of "Digital Systems", University of Piraeus, both for useful guidance and support and also for the valuable knowledge provided throughout the course of my study in the program.

Still, I would like to thank **Stefano Malliaro**[2]**,** for the very nice journey of knowledge, offered me with his valuable and beneficial suggestions.

Last but not least I would like to thank both my classmate and good friend **Georgios Giataganna**s[3] and Mr. **Anastasios Papathanasiou**[4], whose presence was a catalyst for the completion of my diplomatic exercise.

[1] Associate Professor of the Department of Digital Systems, University of Piraeus
[2] Prospective PhD student
[3] Cyber Investigator – Police Lieutenant
[4] Head of Department of Special Financial Cyber Crimes – Police Major

Abstract

This study, both on a theoretical and a practical level, addresses the field of Digital Forensics as far as mobile devices are concerned and specifically the safety of Smartphone devices, which use the operating system of Androids (Mobile Android OS).

Smartphone devices are evolving rapidly and their technology and usability outweigh their predecessors. However their main weakness is security. For the operation of Smartphone devices the activation of defense mechanisms is required in order not to become vulnerable to attempts at tampering or eavesdropping. In addition support measures should be taken in order to improve their existing security.

One of the most basic and most popular Mobile OS operating systems is that of Android, which was selected and is the basis of this study. The operation of both Smartphone devices as well as Android devices poses major safety issues and therefore protection measures both at user level and at the level of application are required.

Moreover, desktop and mobile devices give us the opportunity to recover personal and non personal data from a user. In this case, a series of Mobile Forensics Applications non-commercial are briefly listed and aim at recovering data from Smartphone Devices that run with the operating system Android.

Finally, to better carry out the final part of the research three practical scenarios were applied. The first scenario is associated with Mobile Forensic non commercial application, the second with Mobile Forensic commercial application and the third with Mobile Forensic non-commercial Tool. The basic aim is to recover the credentials (Username & Password) of the user from various applications (applications) of Smartphone device.

1) Introduction

1.1) Context

Today's Mobile devices have transformed from simple voice communication equipment to sophisticated communication devices with the capabilities of computer and far beyond. Smart phones are the devices with the capabilities of a phone and computer. Along with the latest advancements in wireless technologies and the popularity of internet, mobile phones have revolutionized the modern communication in such a way that these devices replaced traditional wallets that people carry without fail. According the to the telecommunication giant Ericsson global mobile penetration reached 85 per cent (%) in 4th Quarter of the year 2011 and mobile subscriptions now total around 6 billion (Ericsson, 2012).

New technologies and innovations have led new intelligent Smartphones with different operating systems capable of doing multiple complex tasks. One such mobile platform is Android Operating system. Android is an open source mobile device platform managed by Open Handset Alliance (OHA). OHA is a consortium led by Google and other industry leaders to develop open standards for mobile devices. Android-based smart phones became so popular among the mobile users in a short span of time and it already positioned as the largest market share in the mobile operating system market (Gartner, 2011).

Mobile phones become a very useful device in people pocket but at the same time it also became a tool for criminal activities. The newer features like location aware, GPS, Bluetooth hacking becomes easier way for criminals to reach out and launch sophisticated criminal activities. Criminals can remotely install the rootkit on mobile phone operating system and can access all the area of the phone including the touch screen and keypad. For example, organized criminal group can easily re-route a customer's phone call to his bank to crime gangs own spoofed call centre and can have access to the victim's bank details and could use it to get away with the all money from the account. According to noted security expert Bruce Schneier, a mobile phone can be turned into a microphone and transmitter for the purpose of listening to conversation of the phone (Schneier, 2006). This indicates that the criminals can remotely access the user's phone and they can operate and continue to maintain their criminal activities as long as they want.

Android forensic is part of digital forensic offers many opportunities and challenges. By looking at the volume of newer android devices and the misuse of these devices by criminals presented challenges how effectively extract and analyze the data for forensic purposes. A very detailed understanding of both the platform and forensic tools are required for acquisition and analysis of mobile data. There are numerous tools on making smart phone forensics easier. This research project evaluates the leading tools available in the market that support Android devices against its ability to extract and analyze the data on multiple parameters.

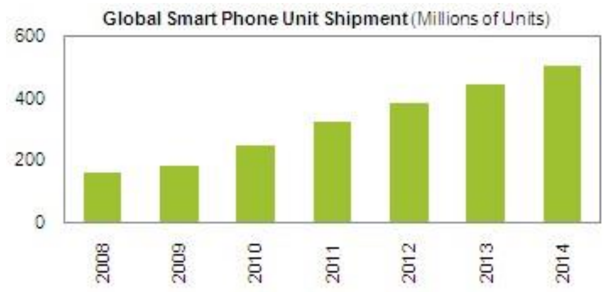


Figure 1 Global Smartphone Unit Shipment

One of the key techniques that are used in data analysis is data acquisition. File acquisition is the process of reassembling data file fragments by using known file structures, heuristics, and other available information. This process allows investigators to recover data that might be "deleted" from a device or from just pieces of data recovered. This is possible because when files are deleted from a device the physical bits of information are not immediately erased/overwritten but only the record of them is deleted.

1.2) Aim & Objective

There are many tools both commercial and open source for the smart phone forensic evaluation, but there is no standard method for the analysis process. It is crucial for these softwares should pose a minimum level of effectiveness when dealing with a forensic analysis. By evaluating these tools it is possible to find the effectiveness of these tools with various parameters. The main aim of the project is to evaluate the extraction tools that are advertised to support Android phones. The objectives to support the aim are:

- ✓ To investigate and review the literature in the field of Mobile device forensic specific to Android forensic
- ✓ Identify and review the best practices currently in use for mobile forensics
- ✓ Determine the measurable criteria and desired outcomes required of Android Forensic tools
- ✓ Enhancement and explicit listing of several useful Android Forensic tools.
- ✓ Develop a methodology for data acquisition from android devices
- ✓ Choosing and practicing specific Android Forensic tools and analysis of it's results.

1.3) Background

There are numerous tools available for the digital forensic that involves computers and other digital devices and become industry standard such as EnCase and FTK. But these tools lack the ability to extract the data from the mobile devices as mobile data is highly volatile and it is often easily altered. Data is information that has some value and these data holds lots of evidential value for a forensic investigator. Till now there is no standard method on how the data can be extracted from mobile devices. Forensic investigators face enormous difficulties when dealing with mobile-based crimes because of its intrusive nature. Data in the mobile phone can be easily modified or even completely wiped out by people who have some knowledge. It is also challenging that these mobile phones stores more data that is well enough to prove the criminal activity originating from the mobile phones.

1.4) Structure of Thesis

Chapter one makes a general overview about new mobile technologies. It points out the positive and negative elements, but also gives emphasis on the value of the operation of Forensic tools (data analysis) targeting always on Android software (Android Forensic tools).

Chapter two refers to the revolution brought by the wave of smartphone devices in the telecommunications sector and the wider section of technology. Additionally extensive reference is made to the value of Mobile Forensics (basic part of Digital Forensics). Particular attention is given to security which structured the whole architecture of Smartphone devices and methods of improvement.

Chapter three deepens in the operating system of Android and describes key features of the background. It even focuses on an essential part of security underlying the whole idea of Android software, presenting together its weaknesses and the potential of improvement.

Chapter five explains the phenomenon of data acquisition and highlights the importance of both the piece of Smartphone devices and the piece of Android software. After analyzing the methodology of the phenomenon (data acquisition) and presenting the value and appropriate cases of extraction tools (either Mobile Forensics applications or Mobile Forensics tools).

Chapter six includes the implementation of the technical section of diplomatic exercise. It describes more fully the preparation of our Smartphone (Samsung Galaxy S GT19000) for the process of extraction data. Then it explains step by step the practical implementation of three Mobile Forensic tools (Root Browser application, Oxygen Forensic Suite {2014} and Linux Memory Extractor Tool {LiME}).

2) Mobile Devices

Overview

Mobile phones have become an integral part of peoples' day-to-day life. Mobiles are used in all sorts of communications such as making calls, sending text messages, sending emails, connecting with friends and family through different social network or instant messaging applications. Mobile phone usage is not limited to basic communication but also heavily used in mobile banking, airline check-in, buy/sell products from various online auction sites, navigating the location, watching movies or videos real time and many other features. It was simply impossible to think the explosive growth of these intelligent devices few years back. There are also different models of mobile devices and sub classifications like PDAs, handheld devices, eBook readers and Smartphones etc.

Android is based on Smartphones that are hybrid devices capable of doing the job of a mobile phone and a computer but always in a portable way. The explosive growth of newer intelligent smart phones based on Android (and IOS) platforms are also initiated brand new methods of criminal activities. These devices carry large amount of data, which are, not limited to just call logs or text messages, but information regarding many other aspects of usage, behavior or other activities. The researches focused in the area of mobile device forensic increased momentum by seeing the value of data stored in these mobile phones.

Figure 2 Mobile Devices

2.1) Revolutionary devices

As it was mentioned above, mobile phones are central to the lives of most people in developed countries and are growing in importance in less developed countries. Since their mainstream adoption in the 1990s, they have remained primarily communication devices. We use mobile phones to talk to other people and we carry mobile phones with us so that other people can talk to us.

However, the situation is changing. Mobile phone manufacturers have developed mobile devices that can serve many functions beyond voice communication such as taking photos and listening to music. Mobile network operators are offering services that give greater value to subscribers, such as portable email for business users. Mobile phones are now equipped with cameras with the potential to turn them into portable bar code scanners. Handset manufacturers are developing RFID chips that can turn mobile phones into mobile wallets able to carry and exchange electronic money securely and engage in other transactions with RFID readers in the physical world.

The combination of more powerful mobile devices, more innovative mobile operators and change in the mobile network infrastructure (such as 3G networks able to carry large amounts of data at high speed as broadband connections do for computers) is setting the stage for an enormous change in a already fast-moving sector. Mobile devices are fast becoming the place where numerous technologies meet and create applications that are useful for both consumers and businesses across the globe. The mobile phone of the future is a device that enables users to communicate, connect, transact and innovate. In most markets, phones with the characteristics below are already becoming available:

- ***A communicative device***

The mobile phone will continue to be a device that is used to communicate with others. Although this may be extended beyond voice to instant messaging and email, it is important not to forget communication is a central strength of mobile devices. As it becomes easier and cheaper to transfer larger amounts of data, sharing photos and videos with others will further extend this role.

- ***A connective device***

Mobile phones enable people to connect to other sources of data anytime, anywhere. This is what is happening with mobile email. As data on the web becomes more structured, mobile devices will become more and more powerful as entry points to tasks that have moved from offline to online but are currently still only available through fixed computers.

- ***A transactional device***

Mobile phones are ideal devices to be used for payments and transactions. There are a wide range of applications that aim to transform the mobile phone into an electronic wallet that can be used as a payment device.

- ***An intelligent device***

Mobile phones are a place where multiple applications can meet and fuse. Mobile devices that integrate a phone, a camera, a location finder (GPS) and a connection to the internet make it possible for a user to request context-dependent information such as finding out where a store selling a product they want to buy is located. As usage increases, mobile phones can become agents of change, tools that facilitate connecting things in the physical world to information about them in the digital world.

2.2) Mobile Forensics

"Mobile phone forensics is the science of recovering digital evidence from a mobile phone under forensically sound conditions using accepted methods"
(Wayne Jansen, Guidelines on Cell Phone Forensics, 2007).

There are a number of research have been conducted in the field of Mobile forensics and some are related to Mobile forensic in general and some are specific to Android and iPhone Forensics. Forensic can be done in many ways on GSM phones but this report mainly focused on the area of Android forensics. The growing number of feature rich phones makes it difficult to create a single forensic tool or standards specific to one platform. Digital evidence in Mobile device is easily susceptible as newer data can be easily overwritten or remote commands it receive from the wireless network. Mobile phones use flash memory to store data. The advantage of flash memory is that it can sustain against impact, high temperature and pressure and making it more difficult to destroy. From a forensic perspective this is good as they can contain deleted information even after an individual attempted to destroy the evidence. The write life cycle of a flash memory is limited and data can be erased by block by block. Mobile devices erase data only when one block is full and also they are an excellent source of digital evidence because they provide great insights that are not available on other devices. Additionally the personal nature of the device makes it easy to establish the last mile evidence required to tie a device to an individual.

Figure 1. Mobile Forensics

2.3) Types of evidences

Though the penetrations of mobile phones are increasing in large scale outnumbering the PC sales, the mobile forensic is still lags behind the digital forensics. The data acquired from mobile phones continues to be used as potential evidence in civil, criminal cases and even high profile cases. However there is still a lot more to do in the field of mobile forensics as no common framework or a standard exist to acquire and analysis of mobile phone data. Forensic professionals often faced with challenges trying to extract evidence from a mobile device mainly due to the fact the small form factor that makes mobile devices so portable. Android mobile forensic tools and toolkits are still immature in dealing with the advances in mobile phone technology. The toolkits are not independently verified or tested for the precise forensic readiness. The developers of forensic tools are using different methods to gain access to memory on the mobile phones. Because of this most tools are limited to minimal number of handsets supported.

In forensics, one of the area forensic professionals looks for evidence is the memory. If a user upgrades the new version there are chances that the memory will be overwritten losing potential evidence from the phone. The primary task of a forensic analyst is to create an exact copy of the device by using cryptographic hash function. But in the case of a mobile phone the hash values (MD5) tend to change and the integrity of the copy will be in a question. The hash values will change each time when the mobile is switched on or off.

Mobile phones now days have better connectivity options that provides whole new world of communication possibilities. It also closely integrated into people lives in such a way that a person may virtually cut off from his peers or from the rest of the world if he missed this gadget. The traditional crimes are migrating to mobile phones and because of its sophisticated nature the criminals can easily modify or even simply wipe out every piece of information poses great challenges for forensic community.

2.4) Forensic Best Practices

Forensic analysts employ different techniques to extract the data out of the device like logical and physical techniques. There are a number of ways that will benefit from the results of an Android forensic investigation. But each may require different procedure and documentation. The first situation is that people think of in general investigations that are to be presented in civil court or criminal court of law. There are internal investigations in corporations that may end up litigated in courts and often used to determine the main cause of an issue. Countries now days face the hack attacks or other malicious hacktivism that threaten the very existence of their citizens well being. Forensic can play a critical role in thwarting attacks against a country by providing a valuable intelligence needed for their governments.

Mobile forensic also involves the same methods of the normal forensic investigation. There are some best practices that need to be followed. Though there are not much standardized format of investigation in mobile forensic the methods of investigation involved is more or less same as digital investigation.

The phases of investigation processes that normally follow are (Wayne Jansen, Guidelines on Cell Phone Forensics, 2007):

• ***Collection***: This is the first and foremost step involved in the investigation. The main purpose here is to collect the potential sources of evidences like mobile, SIM card and other accessories.

• ***Identification***: This is focused on the recognition by labeling the potential sources of digital evidence.

• ***Acquisition***: This is mainly involved in the extraction of data or potential evidence from different sources that have been captured.

• ***Preservation***: One of the important steps involved in the investigation is preservation of evidence where adequate measures should be taken to secure the integrity of the evidence.

• ***Examination*** and ***Analysis***: It involves searching, filtering, examining and evaluation of evidence.

• ***Reporting***: As like any digital investigation reporting is the final part of documented proof of conclusive evidence.

Forensic investigation is a complex process where each and every step of the forensic analyst is very crucial for the preservation of evidence from the target device whether it is a computer or a mobile phone. But having its own removable storage on the computer it is fairly easy to dump or completely remove that specified hardware, but because of its small form factor and the complexity of keeping the hardware in a small area the forensic analyst faces more difficult task in order to acquire the image from the mobile phones. For example in order to acquire the forensic image of a desktop is fairly easy by removing the hardware, connect the physical write blocker and acquire bit-by-bit forensic image of the hard drive. The investigation can take place on the image and the original device would remain unchanged. But again there are high chances of evidence loss while computer is in operation. Even a simple power on or power off will erase the contents of RAM. Mobile devices like Android devices are nearly impossible to forensically analyze without any changes to the device. Unlike desktop or notebook computers the storage part of the Android device cannot be easily removed and if the device is powered on or shutdown of the device again changes the device.

As different forensic examiners approach differently how the data must be acquired there are clearly some deep divide in the forensic community. The Association of Chief Police Officers (ACPO) in United Kingdom produces guidelines called as ‘Good Practice Guide for Computer Based Electronic Evidence establishes four principles for mobile phone evidences (ACPO).

• 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.

• 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.

• 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.

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

These guidelines provide a clear advice on how to act when dealing with mobile phone investigation. As in the case of UK ACPO guidelines the United States also had issued their own guidelines for the Cell phone forensics. In the US, these are issued by NIST (National Institute of Standards and Technology). The ACPO guidelines propose strict guidelines on procedures and practices. The NIST publication is however for to improve the field of mobile devices forensics and not a guide for how law enforcement should handle mobile devices during an investigation. But ACPO lacks the guidance in defining how law enforcement should handle mobile devices during investigations.

2.5) Smartphone Devices

What is really a Smartphone device?

A **Smartphone**, or **Smart phone**, is a mobile phone built on a mobile operating system, with more advanced computing capability and connectivity than a feature phone. The first Smartphones combined the functions of a personal digital assistant (PDA), including email functionality, with a mobile phone. Later models added the functionality of portable media players, low-end compact digital cameras, pocket video cameras, and GPS navigation units to form one multi-use device. Many modern Smartphones also include high-resolution touchscreens and web browsers that display standard web pages as well as mobile-optimized sites. High-speed data access is provided by Wi-Fi, mobile broadband, NFC and Bluetooth. In recent years, the rapid developments of mobile app markets and mobile commerce have been drivers of Smartphone adoption.

The mobile operating systems (OS) used by modern Smartphones include Google's Android, Apple's iOS, Nokia's Symbian, Blackberry Ltd's BlackBerry 10, Samsung's Bada, Microsoft's Windows Phone, Hewlett-Packard's webOS, and embedded Linux distributions such as Maemo and MeeGo. Such operating systems can be installed on many different phone models, and typically each device can receive multiple OS software updates over its lifetime. A few other upcoming operating systems are Mozilla's Firefox OS, Canonical Ltd.'s Ubuntu Phone, and Tizen.

Worldwide sales of Smartphones exceeded those of feature phones in early 2013. As of July 18, 2013, 90 percent of global handset sales are attributed to the purchase of Android and iPhone Smartphones.

History

Devices that combined telephony and computing were conceptualized as early as 1973, and were offered for sale beginning in 1994. The term "Smartphone", however, did not appear until 1997, when Ericsson described its GS 88 "Penelope" concept as a *Smart Phone*.

The distinction between Smartphones and feature phones can be vague, and there is no official definition for what constitutes the difference between them. One of the most significant differences is that the advanced application programming interfaces (APIs) on Smartphones for running third-party applications can allow those applications to have better integration with the phone's OS and hardware than is typical with feature phones. In comparison, feature phones more commonly run on proprietary firmware, with third-party software support through platforms such as Java ME or BREW. An additional complication is that the capabilities found in newer feature phones exceed those of older phones that had once been promoted as Smartphones.

Some manufacturers and providers use the term "superphone" for their high end phones with unusual large screens and other expensive features. With the advent of devices with larger screens, the term "phablet", a portmanteau of the words *phone* and *tablet*, had come into common usage by 2008.

The first cellular phone to incorporate PDA features was an IBM prototype developed in 1992 and demonstrated that year at the COMDEX computer industry trade show. A refined version of the product was marketed to consumers on 16 August 1994 by BellSouth under the name Simon Personal Communicator. The Simon was the first device that can be properly referred to as a "Smartphone", even though that term was not yet coined. In addition to its ability to make and receive cellular phone calls, Simon was also able to send and receive facsimiles, e-mails and pages through its touch screen display. Simon included many applications including an address book, calendar, appointment scheduler, calculator, world time clock, games, electronic note pad, handwritten annotations and standard and predictive touch-screen keyboards.

In 1996, Nokia released the Nokia 9000, part of the Nokia Communicator line, which became their best-selling phone of that time. It was a palmtop computer-style phone combined with a PDA from HP. In early prototypes, the two devices were fixed together via a hinge in what became known as a clamshell design. When opened, the display of 640×200 pixels was on the inside top surface and with a physical QWERTY keyboard on the bottom. Email and text-based web browsing was provided via the GEOS V3.0 operating system.

In the late 1990s though, the vast majority of mobile phones had only basic phone features so many people also carried a separate dedicated PDA device, running early versions of operating systems such as Palm OS, BlackBerry OS or Windows CE/Pocket PC. These operating systems would later evolve into mobile operating systems and power some of the high-end Smartphones.

In early 2001, Palm, Inc. introduced the Kyocera 6035. This device combined a PDA with a mobile phone and operated on the Verizon Wireless network. It also supported limited web browsing. The device was not adopted widely outside North America. Between 2002 and 2004 HTC gained much popularity in Europe with their "Wallaby", "Falcon" and "Himalaya" models running the Windows Mobile "Pocket PC" Operating System. In 2004, HP released the iPaq h6315, a device that combined their previous PDA, the HP 2215 with cellular capability.

In the wake of controversy regarding the sourcing of materials and Smartphone manufacturing process, the Fairphone company launched its first "socially ethical" Smartphone at the London Design Festival in 2013. The company places an emphasis on changing the manner in which supply chains and commercial strategies work, and the first shipment of Fairphone handsets will occur in December 2013.

The QSAlpha commenced production of the Quasar IV—a Smartphone designed entirely around security, encryption and identity protection—in partnership with a large consumer electronics OEMS manufacturer in late 2013. The handset uses specialized technology to safely protect all incoming and outgoing communications, and is targeted at customers who require incredibly high levels of security and encryption.

Operating systems (OS)

Android

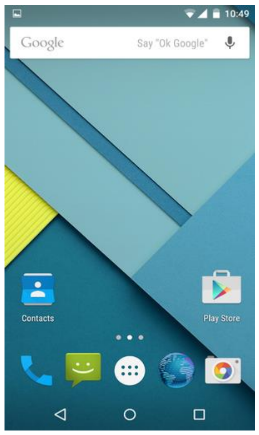


Figure 3 Android 5.0 (Lollipop)

Android[5] is a mobile operating system (OS) based on the Linux kernel and currently developed byGoogle. With a user interface based on direct manipulation, Android is designed primarily fortouchscreen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen

[5] https://www.android.com/

input, it also has been used in game consoles, digital cameras, regular PCs (e.g. the HP Slate 21) and other electronics.

As of July 2013 the Google Play store has had over one million Android applications ("apps") published, and over 50 billion applications downloaded. A developer survey conducted in April–May 2013 found that 71% of mobile developers develop for Android. At Google I/O 2014, the company revealed that there were over one billion active monthly Android users, up from 538 million in June 2013. As of 2015, Android has the largest installed base of all general-purpose operating systems.

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software, including proprietary software developed and licensed by Google. Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance—a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.

Android is popular with technology companies which require a ready-made, low-cost and customizable operating system for high-tech devices. Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially released running other operating systems. The operating system's success has made it a target for patent litigation as part of the so-called "smartphone wars" between technology companies.

iOS

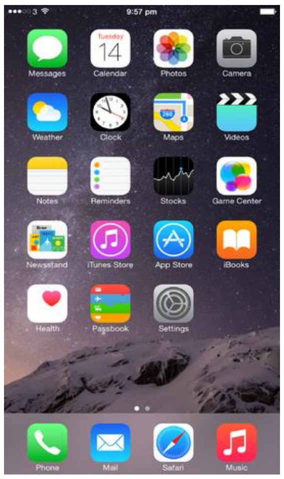


Figure 4 iOS 8 running on an iPhone

iOS[6] (originally **iPhone OS**) is a mobile operating system developed by Apple Inc. and distributed exclusively for Apple hardware. It is the operating system that powers many of the company's iDevices.

Originally unveiled in 2007 for the iPhone, it has been extended to support other Apple devices such as the iPod Touch (September 2007), iPad (January 2010), iPad Mini (November 2012) and second-generation Apple TVonward (September 2010). As of June 2014, Apple's App Store contained more than 1.2 million iOS applications, 500,000 of which were optimized for iPad.[9][10] These apps have collectively been downloaded more than 60 billion times. It had a 21% share of the smartphone mobile operating system units shipped in the fourth quarter of 2012, behind Google's Android. By the middle of 2012, there were 410 million devices activated. According to the special media event held by Apple on September 12, 2012, 400 million devices had been sold by June 2012.

The user interface of iOS is based on the concept of direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. Interaction with the OS includes gestures such as*swipe*, *tap*, *pinch*, and *reverse pinch*, all of which have specific definitions within the context of the iOS operating system and its multi-touch interface. Internal accelerometers are used by some applications to respond to shaking the device (one common result is the undo command) or rotating it in three dimensions (one common result is switching from portrait to landscape mode).

iOS shares with OS X some frameworks such as Core Foundation and Foundation; however, its UI toolkit isCocoa Touch rather than OS X's Cocoa, so that it provides the UIKit framework rather than the AppKitframework. It is therefore not compatible with OS X for applications. Also while iOS also shares the Darwinfoundation with OS X, Unix-like shell access is not available for users and restricted for apps, making iOS not fully Unix-compatible either.

Major versions of iOS are released annually. The current release, iOS 8.1.3, was released on January 27, 2015. In iOS, there are four abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer. The current version of the operating system (iOS 8.0), dedicates 1.3 - 1.5GB of the device's flash memory for the system partition, using roughly 800 MB of that partition (varying by model) for iOS itself. It runs on the iPhone 4S and later, iPad 2 and later, all models of the iPad Mini, and the 5th-generation iPod Touch.

[6] https://www.apple.com/ios/

Windows Mobile

Figure 5 Windows Mobile 6.5 screenshot

Windows Mobile[7] is a Smartphone OS developed and maintained by Microsoft. The OS's worldwide Smartphone market share decreased from 7.9% in the Q3 of 2009 to 2.8% in the Q3 of 2010 (Gartner, 2010). The latest version of the OS is Windows Phone 8. Therefore, in this section we present the security model of Windows Mobile 6, since its development API is available.

The security model of Windows Mobile (Microsoft, 2010c) depends on the enabled policy of the device. This policy is responsible for controlling which applications are allowed to be executed on the device, what functionality of the OS is accessible to the application, how desktop applications interact with the Smartphone, and how the user or application access specific device settings. The enabled policy on a Windows Mobile Smartphone is either one-tier access or two-tier access (Microsoft, 2010c).

A device with one-tier access policy enabled, only controls if one application runs on the device or not, without inspecting if the application is using sensitive API. This decision depends on whether the application's installation package file (.cab file) is correctly signed with a certificate that exists in the device's certificate store. If the application is signed with a known certificate, then the application runs in privileged mode, with the ability to call any API, access and modify anything in the device's file system and registry. Otherwise, if the application is unsigned or signed with a certificate that is not known, further policy checks take place for the decision of application execution. In this case, security policies settings define whether the user is prompted to give her consent for the application to run. It must be clarified that if the user permits the execution, then the application will run in privileged mode. This means that an unknown and unsigned application maintains full access to the device.

On the other hand, a device with two-tier access policy enabled, apart from controlling application execution, it also checks runtime permissions by controlling the APIs that the application uses. Access to protected API is determined by the application's digital signature. More specifically, if the application is

[7] http://en.wikipedia.org/wiki/Windows_Mobile

signed with a known certificate (i.e. a certificate present in the device's certificate store), then the application is executed without further checks, granted the permissions defined by the certificate class.

In the case that the certificate belongs to the Privileged Execution Trust Authorities certificate store, the application is executed with privileged permissions. Otherwise, the application is executed in normal mode. When the application is unsigned or signed with an unknown certificate, then further checks are required to determine if the application is allowed to run in normal mode. It is worth noting that the functionality provided by normal privileges is enough for most third-party developed applications.

According to the default security configuration of Windows Mobile, provides weak security protection as: a) it allows the execution of unsigned applications or singed ones with an unknown certificate, b) in case the user is prompted to authorize the execution of the application. Hence, in both access tiers of the default security configurations, unsigned and unknown code is executed with the user's approval either in normal mode (two-tier access) or privileged mode (one-tier access). Furthermore, although one tier access does not provide strong security, it is the default access tier in some devices (Microsoft, 2010c). Nonetheless, it should be noted that the security model permits Mobile Operators to make post-production changes to security settings configurations of the device.

The security model of Windows Mobile includes security mechanisms enabling a Mobile Operator to revoke (i.e. remove) applications running on Smartphones (Microsoft, 2010c). The revocation may concern either a) a class of applications signed with the same certificate, where the corresponding certificate is being revoked, or b) a specific application binary, where the hash of the binary is being transferred to the device with transfer mechanisms described in (Microsoft, 2010c).

For application implementation in Windows Mobile 6, Microsoft freely provides the required development toolkit (i.e. SDK, emulator, documentation, etc.). The supported implementation languages (e.g. C#, Visual C++) are compatible with the Compact .NET Framework.

For the acquisition of certificates that are known to the devices, the developer opts from the paid services provided by Microsoft (Microsoft, 2011b).

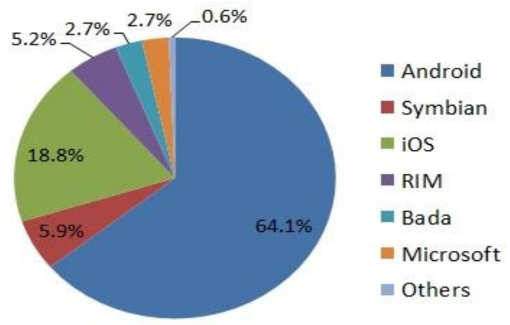


Figure 6 Statistical Graph of Mobile OS

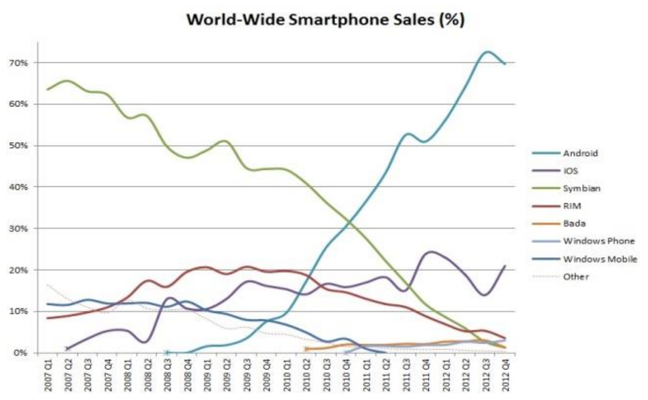


Figure 7 Statistical Graph of World Smartphone Sales

Windows Phone

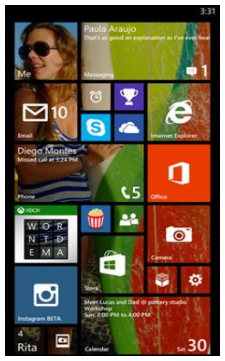


Figure 8 Windows Phone 8.1 Screenshot

Windows Phone[8] (WP) is a family of mobile operating systems developed by Microsoft for smartphones as the replacement successor to Windows Mobile and Zune. Windows Phone features a new user interface derived from Metro design language. Unlike Windows Mobile, it is primarily aimed at the consumer market rather than the enterprise market. It was first launched in October 2010 with Windows Phone 7.

Windows Phone 8.1 is the latest public release of the operating system, released to manufacturing on April 14, 2014.

The Windows Phone brand was phased out in accordance to Microsoft's new branding strategy and is due to be succeeded in 2015 with the release of Windows 10 (mobile), which is designed to provide a universal experience in conjunction with the PC version of Windows 10.

[8] http://www.windowsphone.com/en-us

BlackBerry

Figure 9 BlackBerry Z10

BlackBerry[9] is a line of wireless handheld devices (commonly called smartphones) and services designed and marketed by BlackBerry Limited, formerly known as Research In Motion Limited (RIM). The very first RIM device was the Inter@ctive Pager 900, a clamshell type device that allowed two way paging, announced on September 18, 1996. After the success of the 900 the Inter@ctive Pager 800 was created for IBM who bought 10 Million dollars of them on February 4, 1998.[3] The next device to be released was the Inter@ctive Pager 950 on August 26, 1998. The very first device to carry the BlackBerry name was the BlackBerry 850, an email pager, released January 19, 1999. Although identical looking to the 950, the 850 was the first device to integrate email and the name Inter@ctive Pager was no longer used to brand the device.

The most recent BlackBerry devices are the BlackBerry Passport, BlackBerry Classic, and BlackBerry Z30. Theuser interface varies by model; most had featured a physical QWERTY keyboard, while newer generations have relied on a multi-touch screen and virtual keyboard.

BlackBerry devices can record video, take photos, play music and also provide functions such as web-browsing, email messaging, instant messaging, and the multi-platform BlackBerry Messenger service.

It was one of the major smartphone vendors until 2012. The consumer BlackBerry Internet Service is available in 91 countries worldwide on over 500 mobile service operators using various mobile technologies.[4] As of September 2013, there were 85 million BlackBerry subscribers worldwide. That number has dropped steadily, down to 46 million in June 2014.

9 http://global.blackberry.com/sites.html?

Bada

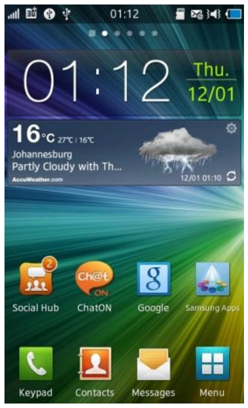


Figure 10 Bada 2.0 Home screen

Bada[10] was an operating system for mobile devices such as smartphones andtablet computers. It was developed by Samsung Electronics. Its name means "ocean" or "sea" in Korean. It ranges from mid- to high-end smartphones.

To foster adoption of Bada OS, since 2011 Samsung reportedly has considered releasing the source code under an open-source license, and expanding device support to include Smart TVs. Samsung announced in June 2012 intentions to merge Bada into the Tizen project, but would meanwhile use its own Bada operating system, in parallel with Google Android OS and Microsoft Windows Phone, for its smartphones.

All Bada-powered devices are branded under the *Wave* name, but not all of Samsung's Android-powered devices are branded under the name *Galaxy*.

On 25 February 2013, Samsung announced that it will stop developing Bada, moving development to Tizeninstead. Bug reporting was finally terminated in April 2014.

[10] http://developer.bada.com/apis/index.do

Symbian

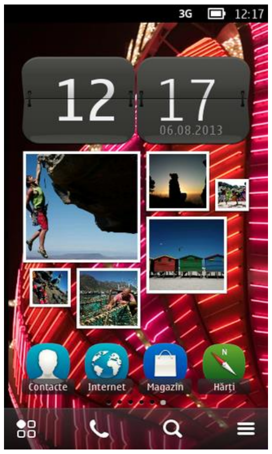


Figure 11 Home Screen of Nokia Belle Feature Pack 2 (Latest version of Symbian)

Symbian[11] was a closed-source mobile operating system (OS) and computing platform designed forsmartphones. Symbian was originally developed by Symbian Ltd., as a descendant of Psion's EPOC and runs exclusively on ARM processors, although an unreleased x86 port existed. The current form of Symbian is an open-source platform developed by Symbian Foundation in 2009, as the successor of the original **Symbian OS**. Symbian was used by many major mobile phone brands, like Samsung, Motorola, Sony Ericsson, and above all by Nokia. It was the most popular smartphone OS on a worldwide average until the end of 2010, when it was overtaken by Android.

Symbian rose to fame from its use with the S60 platform built by Nokia, first released in 2002 and powering most Nokia smartphones. UIQ, another Symbian platform, ran in parallel, but these two platforms were not compatible with each other. Symbian, was officially released in Q4 2010 as the successor of S60 and UIQ, first used in theNokia N8, to use a single platform for the OS. In May 2011 an update, Symbian Anna, was officially announced, followed by Nokia Belle (previously Symbian Belle) in August 2011.

On 11 February 2011, Nokia announced that it would use Microsoft's Windows Phone OS as its primary smartphone platform, whilst Symbian would be gradually wound down. On 22 June 2011 Nokia signed an agreement for Accenture to provide Symbian-based software development and support services to Nokia through 2016; about 2,800 Nokia employees became Accenture employees as of October 2011. The transfer was completed on 30 September 2011. The Nokia 808 PureView is officially the last Symbian smartphone. In January 2014 Nokia stopped accepting new or changed Symbian software from developers, effectively terminating its support of the operating system.

[11] http://en.wikipedia.org/wiki/Symbian

Palm OS

Figure 12 Palm m505, running Palm OS 4.0

Palm OS[12] (also known as **Garnet OS**) is a mobile operating system initially developed by Palm, Inc., forpersonal digital assistants (PDAs) in 1996. Palm OS was designed for ease of use with a touchscreen-basedgraphical user interface. It is provided with a suite of basic applications for personal information management. Later versions of the OS have been extended to support smartphones. Several other licensees have manufactured devices powered by Palm OS.

Following Palm's purchase of the Palm trademark, the currently licensed version from ACCESS was renamed*Garnet OS*. In 2007, ACCESS introduced the successor to Garnet OS, called Access Linux Platform and in 2009, the main licensee of Palm OS, Palm, Inc., switched from Palm OS to webOS for their forthcoming devices.

[12] http://gl.access-company.com/

Ubuntu Touch

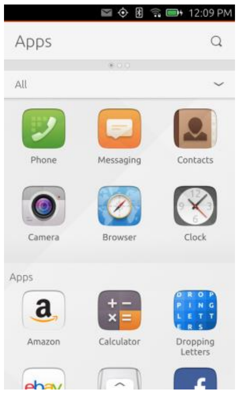


Figure 13 The Ubuntu Touch home screen showing applications

Ubuntu Touch[13] (also known as **Ubuntu Phone**) is a mobile version of the Ubuntu operating system developed by Canonical UK Ltd and Ubuntu Community. It is designed primarily for touchscreen mobile devices such assmartphones and tablet computers.

Firefox OS

Figure 14 Firefox OS 2.1 nightly build lock screen

13 http://www.ubuntu.com/phone

Firefox OS[14] (project name: *Boot to Gecko*, also known as *B2G*) is a Linux kernel-based open-sourceoperating system for smartphones and tablet computers[6] and is set to be used on smart TVs. It is being developed by Mozilla, the non-profit organization best known for the Firefox web browser.

Firefox OS is designed to provide a complete, community-based alternative system for mobile devices, using open standards and approaches such as HTML5 applications, JavaScript, a robust privilege model,open web APIs to communicate directly with cellphone hardware, and application marketplace. As such, it competes with commercially developed operating systems such as Apple's iOS, Google's Android,Microsoft's Windows Phone and Jolla's Sailfish OS.

Firefox OS was publicly demonstrated in February 2012, on Android-compatible smartphones. In January 2013, at CES 2013, ZTE confirmed they would be shipping a smartphone with Firefox OS,[11] and on July 2, 2013, Telefónica launched the first commercial Firefox OS based phone, ZTE Open, in Spain which was quickly followed by GeeksPhone's Peak+. As of December 16, 2014, Firefox OS phones are offered by 14 operators in 28 countries throughout the world.[15]

Mozilla has also partnered with T2Mobile to make a Firefox OS reference phone dubbed "Flame" which is designed for developers to contribute to Firefox OS and to test apps.

Sailfish

Sailfish

Figure 15 Sailfish version 1.1.2.16

Sailfish[15] is a mobile operating system (OS) combining the Linux kernel, the Mer core and proprietary software written by mobile software developer Jolla. Sailfish is being developed by Jolla in cooperation with the Mer project community and corporate members of the Sailfish Alliance. Sailfish is used in the Jolla smartphone, in the upcoming Jolla Tablet, and by other licensees. The OS is mainly targeted at mobile devices and is also intended to support other devices.

[14] https://www.mozilla.org/firefox/os
[15] https://sailfishos.org/

Tizen

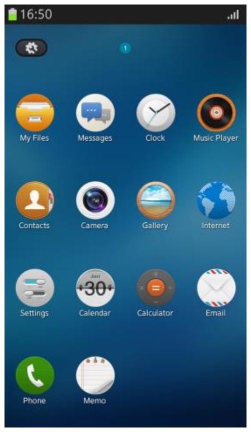


Figure 16 Tizen 2.2 beta screen

Tizen[16] (/ˈtaɪzɛn/) is an operating system based on the Linux kernel and the GNU C Library implementing theLinux API. It targets a very wide range of devices including smartphones, tablets, in-vehicle infotainment (IVI) devices, smart TVs, PCs, smart cameras, wearable computing (such as smartwatches), Blu-ray players, printersand smart home appliances (such as refrigerators, lighting, washing machines, air conditioners,ovens/microwaves and a robotic vacuum cleaner). Its purpose is to offer a consistent user experience across devices. Tizen is a project within the Linux Foundation and is governed by a Technical Steering Group (TSG) composed of Samsung and Intel among others.

Tizen's licensing model involves software that uses a variety of licenses that may be incompatible with the Open Source Definition (see Licensing model below), and a proprietary software development kit (SDK).

The Tizen Association was formed to guide the industry role of Tizen, including requirements gathering, identifying and facilitating service models, and overall industry marketing and education. Members of the Tizen Association represent every major sector of the mobility industry and every region of the world. Current members include telecommunications network operators, OEMs and manufacturers: Fujitsu, Huawei, Intel, KT Corporation, NEC Casio Mobile Communications, NTT DoCoMo, Orange S.A., Panasonic Mobile Communications, Samsung, SK Telecom, Sprint Corporation and Vodafone. While the Tizen Association decides what needs to be done in Tizen, the Technical Steering Group determines what code is actually incorporated into the operating system to

[16] https://www.tizen.org/

accomplish those goals. Tizen roots back to the Samsung Linux Platform (SLP) and the LiMo Project and in 2013 Samsung merged its homegrown Bada project into Tizen.

The first week of October 2013, Samsung's NX300M smart camera became the first consumer product based on Tizen; it was sold in South Korea for a month before its OS was revealed at the Tizen Developer Summit,[8][9][10]then became available for pre-order in the United States in early 2014 with a release date of March 1. The first Tizen tablet was announced by Systena in June 2013, a 10-inch quad-core ARM with 1920x1200 resolution that was eventually shipped in late October 2013 as part of a development kit exclusive to Japan. TheSamsung Gear 2 smartwatch uses Tizen and it was released in April 2014.[14] The Samsung ZEQ 9000 was expected to be the first commercially available smartphone running the operating system, but its planned launch at Mobile World Congress 2014 did not happen.

The ZEQ 9000 Tizen smartphone was postponed, prompting The Wall Street Journal to call Tizen "an ill-fated project". Samsung released the Tizen-based Samsung Z1 to the Indian market in January 2015.

2.5.1) Smartphone security

Overview

Mobile devices present an attractive target to cyber-criminals. They are used everywhere, contain a vast amount of personal and confidential information and can be used to perform all kinds of online transactions.

A common Smartphone security concern is their communication channels. In this sense, they are more vulnerable than traditional PCs and can be subjected to various attack vectors –SMS, Bluetooth, Wi-Fi, Web browsers, applications and email an aspect that can result in the proliferation of malicious code targeting these platforms.

Cell phones are really personal devices and it is precisely this personalization that makes them even more unsafe. We might have one computer for a family but every family member has a personal device and it is with them all the time. Contributing to their security risks is that many users are unaware they can be a security hazard and that battery limitations of the devices themselves prevent running complex applications like antivirus solutions.

Figure 17 Smartphone Security

2.5.1.1) False sense of security.

From a purely physical point of view, Smartphones feel like very personal devices. We carry them around with you and control their operation, which can make you believe they are less accessible to intruders. This false sense of security, combined with phones often linking to personal email applications, social networks and multimedia content, can lead to private and confidential information being stored, sometimes inadvertently.

This false sense of security can sometimes make users overlook basic precautions such as changing default device security settings.

At present, the number of Smartphone attacks is quite small compared to attacks on PCs. There are more than 60 million known malicious programs for PCs as opposed to 600 for Smartphones, although we expect an increase in the amount of malicious code targeting the latter.

With regard to this, on September 25, 2010, S21sec reported the first malware strain capable of bypassing the two-factor (PC client and cell phone) authentication systems used to enhance online banking security. These systems are discussed at a later point in this report.

There are many factors that will contribute to the proliferation of new threats. The only security mechanism implemented in most Smartphones is a password, making the security and reliability of downloaded applications integral to Smartphone safety.

It is advisable to have encryption mechanisms to block access to lost or stolen devices. This is particularly important in organizations and companies where there is a thin line between corporate security policies and personal use due to the popularity of these devices among the general public.

Smartphone security must not only be considered from the end user‘s point of view. It involves many other aspects like the Smartphone‘s operating system kernel, application deployment and platform development environments.

Figure 18 False sense of Security

2.5.1.2) Security issues.

Data leakage resulting from device loss or theft

The Smartphone is stolen or lost and its memory or removable media are unprotected, allowing an attacker access to the data stored on it. Smartphones, being both valuable and pocket-sized, are likely to be stolen or lost. In a recent UK government survey, 2% reported their mobile phone was stolen last year. If data on the Smartphone memory or its removable media is not sufficiently protected (by encryption) then an attacker can access that data.

Smartphones often contain valuable information such as credit card data, bank account numbers, passwords, contact data, and so on. They are often the user's primary repository of personal data because they are carried around all the time and are always available. Users sometimes protect sensitive information by storing it in an obfuscated form (see example below). Business phones often contain corporate emails and documents and may contain sensitive data. Even though encryption is implemented, weaknesses may exist in the implementation of encryption in Smartphones. This is not to suggest that encryption is not recommended, but to encourage caution in selecting the solution used.

Unintentional disclosure of data

The Smartphone user unintentionally discloses data on the Smartphone. Users are not always aware of all the functionality of Smartphone apps. Even if they have given explicit consent, users may be unaware that an app collects and publishes personal data trace users and so allow, for example, stalking, robbery or the hijacking of trucks containing valuable goods.

A fundamental underlying vulnerability is the difficulty of collecting meaningful consent for the processing of all the personal data available on a Smartphone. Certain types of data collection naturally lend themselves to integration with user consent, without having to assume the persistence of a decision. For example, file upload involves the user in selecting the file and thus giving consent (to that file being uploaded) as an integral part of the process. Other types of data are more problematic and location data is a good example, as it is not feasible for the user to have to consent every time a new location is disclosed. Location data, for example, is often used in social networks – in messages or uploaded photo metadata, in augmented reality apps, micro-blogging posts, etc. Most apps have privacy settings for controlling how and when location data is transmitted, but many users are unaware (or do not recall) that the data is being transmitted, let alone know of the existence of the privacy setting to prevent this.

For example the location of data is often included in image files. Users, by giving an application access to the image files, may be unintentionally disclosing their whereabouts.

Attacks on decommissioned Smartphones

The Smartphone is decommissioned improperly allowing an attacker access to the data on the device. Due to a growing awareness of identity theft many people and organizations now destroy or wipe computer hard drives before decommissioning. However, the same thing is not yet happening with Smartphones. At the same time, more and more devices are being recycled. According to market analysts ABI Research, by 2012 over 100 million mobile phones will be recycled for reuse each year. As previously mentioned, Smartphones contain large amounts of sensitive information which may be valuable to an attacker. They are an increasingly attractive target for Smartphone dumpster divers.

For example in a recent study, mobile phones were bought second-hand on eBay and, out of the 26 business Smartphones, 4 contained information from which the owner could be identified while 7 contained enough data to identify the owner's employer. The research team managed to trace one Smartphone to a senior sales director of a corporation, recovering call history, address book entries, diary, emails, etc.

Phishing attacks

An attacker collects user credentials (such as passwords or credit card numbers) by means of fake apps or (SMS, email) messages that seem genuine. Phishing attacks are a well-known threat for users of traditional PCs. Phishing attacks are actually platform independent, because the attacker does not need to attack the user’s device in any way. However, there are a number of reasons why the risk of phishing is important for Smartphone users:

- ✓ Smartphones have a smaller screen, which means that attackers can more easily disguise trust cues that users rely on to decide on submitting credentials; e.g. cues that show whether the website uses SSL.
- ✓ App-stores provide a new way of phishing by allowing attackers to place fake apps in the app-store, disguising them as legitimate apps.
- ✓ Smartphones provide additional channels that can be used for phishing, e.g. SMS (SMiShing). Users may be less cautious about SMS phishing messages.
- ✓ Smartphones are a new type of device and users may not be aware of the fact that phishing is a risk on Smartphones as well.

Spyware attacks

The Smartphone has spyware installed, allowing an attacker to access or infer personal data. Spyware covers untargeted collection of personal information as opposed to targeted surveillance. It is malicious software that covertly collects information about users and their activities to use it for marketing purposes, such as profiling or targeted advertisements. Such spyware is often apparently bona fide software, installed with the user's consent, which requests and abuses privileges over and above those required for the stated purpose of the app.

The amount of personal data, sensitive documents and credentials stored and processed by Smartphones, makes them an interesting target for spyware. Furthermore, Smartphones provide covert channels through which data may be disclosed (by an application) to an attacker. Even when it seems there is a legitimate need for an app to send data over a particular channel, the permission model of Smartphones is not always granular enough to protect users against abuse. For example, a weather app may ask permission to use location data and to connect to the Internet, which seems legitimate (to get fresh location-based weather data). The app may however abuse this permission by sending location-data to advertisement servers for marketing purposes.

First example: SMobile describes a study of 48,694 applications in the Android market, which found that one in every five applications requests permissions to access private or sensitive information that an attacker could use for malicious purposes. One out of every twenty applications has the ability to place a call to any number without interaction with or authority from the user.

Furthermore, data access by apps is sometimes exempt from explicit user permissions. For example, in iOS, the address book is accessible to all apps. No special status is given to the user's own contact details in the address book, meaning that, apart from the large amounts of personal data this exposes, the user's own phone number is also accessible, which can be used for unsolicited marketing. Another important vulnerability is the fact that on the iPhone the keyboard cache is accessible to all apps; although this does not include sensitive information such as passwords, it does contain a lot of private information.

Second example: The app Jigsaw is able to recognize user activities based on an analysis of microphone, GPS and accelerometer for patterns characteristic of routine activities. For example, the jolts produced when the user is walking depend on whether the phone is in a trouser or jacket pocket, so the software can recognize both patterns. It is designed to minimize the drain on the phone's battery.

Third example: The app Sensor Logger records changes in the phone's accelerometer and magnetic field sensors over a period of 50 seconds and attempts to determine the activity (walking, standing, sitting) in which the user is engaged.

Network Spoofing Attacks

An attacker deploys a rogue network access point (WiFi or GSM) and users connect to it. The attacker subsequently intercepts (or tampers with) the user communication to carry out further attacks such as phishing.

Rogue WiFi hotspots and Bluetooth devices can be used to intercept and tamper with the network communication to the Smartphone. Rogue Internet gateway names may be configured on the Smartphone by a malicious SMS configuration message. In this attack, a spoofed service configuration SMS is used to change the default access point used by the phone. A more complicated spoofing attack relies on mounting a rogue GSM base station. The hardware required to set up such a base station has become relatively inexpensive. This attack is not feasible on 3G networks because of network integrity keys. A rogue WiFi hotspot or other spoofed network nodes can be used as a means to carry out several other attacks, e.g. phishing, SSL downgrade attacks, eavesdropping, etc (making it less likely using 3G networks).

Theoretically speaking, such attacks should be detectable by the user. However, in practice most users do not pay attention to trust cues such as SSL certificates or whether a site uses SSL. For Smartphone users the risk is even higher because security indicators (such as a ‘trusted SSL connection’ indicator) are harder to find or missing on Smartphones. For Smartphone users the risk is even higher because security indicators (such as a ‘trusted SSL connection’ indicator) are harder to find or missing on Smartphones.

For example at the Blackhat 2009 conference a presenter used a rogue WiFi hotspot to mount an SSL downgrade attack and was thus was able to capture 20 email passwords from security professionals.

Surveillance attacks

An attacker keeps a user under surveillance through the user’s own Smartphone. Smartphones can be used to keep a targeted individual under surveillance. They contain multiple sensors such as a microphone, camera, accelerometer and GPS. This, combined with the possibility of installing third-party software and the fact that a Smartphone is closely associated with an individual, makes it a useful spying tool.

Given short-term physical and logical access to a device, it is possible to install comprehensive spying tools on it. Sometimes the user can be tricked into helping the attacker by installing malicious apps. There are also already several examples of legitimate software, whose express purpose is to allow an attacker to keep the mobile user under surveillance. Furthermore, even tools that are not designed for spyware may be configured covertly to allow for tracking.

The GPS sensor deserves particular attention in this regard since it is a source of highly sensitive personal information – e.g. information about when someone is not at home can be useful to burglars. As mentioned above, even a combination of seemingly innocuous sensor data (e.g. magnetic field history) could be used to deduce sensitive information about an individual and their environment.

Example 1: The app Tap Snake, ostensibly a simple snake game, captures GPS location data and uploads it to a remote server.

Diallerware attacks

An attacker steals money from the user by means of malware that makes hidden use of premium SMS services or numbers. Certain Smartphone API calls cost the user money, e.g. SMS (including micropayments), phone calls, and data over metered GSM/UMTS. If an attacker can install an app on the user’s smartphone, which is able to make such API calls covertly or trick the user into giving consent to their use, they can steal money from the Smartphone user. The risk of this attack for consumers is judged as high because they are usually on a more limited budget and are more likely to download rogue apps.

For example scammers are distributing corrupted versions of shareware games for Smartphones which make calls to premium-rate numbers across the globe, racking up expensive bills without the phone owner's knowledge.

Financial malware attacks

The Smartphone is infected with malware specifically designed for stealing credit card numbers, online banking credentials or subverting online banking or ecommerce transactions.

Financial malware is software specifically designed to steal credentials or perform man-in-the-middle attacks on financial applications or web services. Like PCs, Smartphones are also vulnerable to banking malware. It may be a key-logger collecting credit card numbers, or it may be more sophisticated and intercept SMS authentication codes to attack online banking applications. Another strategy is for an attacker to submit an app to an app-store, impersonating a real banking app. If users download and use the app, the attacker can mount a man-in-the-middle attack on banking transactions.

Smartphones have been relatively safeguarded from malware (compared to PCs). This may be due to the efforts from platform vendors or simply because traditional PCs still provide an easier and more interesting target for attackers. Nonetheless, malware for smartphones is a serious risk.

For example ZeuS Mitmo (Man in the Mobile) is an example of an attack that exploits the combined features unique to the Smartphone. ZeuS Mitmo combines the SMS and Web attack vectors to target online banks via the Smartphone.

Network congestion

Network resource overloads due to Smartphone usage leading to network unavailability for the end-user. The uptake of Smartphones and mobile Internet increases the risk of network congestion. Network congestion can occur in two ways:

- Signaling overload: always-on Smartphone apps are constantly polling the network for updated information. For every bit of data sent, a large number of signaling messages are sent (e.g. keep-alive messages). A typical Smartphone generates 8 times more signaling traffic than a laptop with a USB dongle.
- Data capacity overload: Cisco estimates that mobile data traffic will double every year through 2014, increasing 39 times between 2009 and 2014. Mobile data traffic will grow at a compound annual growth rate of 108 percent between 2009 and 2014, reaching 3.6 million terabytes per month by 2014.

To address signaling overload, there are mechanisms that change how often a Smartphone switches between idle and active mode, such as the 3GPP Fast Dormancy mechanism.

In terms of data capacity (as opposed to signaling load), solutions such as LTE and WiMAX promise improvements in spectral efficiency, the amount of data that can be transmitted over the air using the same amount of allocated spectrum. At the same time, however, it has been argued that average data demand per network user will outstrip data capacity by 2013 and there are concerns that 'wireless technology is approaching theoretical limits of spectral efficiency.

In the longer term in Europe, the risk is reduced by the fact that spectrum is being released by the cessation of analogue TV and 2G services, which is likely to be made available for such applications. However, it is worth noting that while on average, this threat may not be very serious, critical events such as natural disasters which cause a sudden peak in demand may be create conditions which put data networks used by Smartphones under severe strain.

Example 1: A widely publicized case was AT&T‟s first introduction of the iPhone, which caused massive disruption of their data network. This seems to be a problem in many EU countries as well. The Italian Telecommunications Authority recently warned of „network collapse‟ due to Smartphone and 3G card usage.

Example 2: There was also a complete loss of data connectivity at Microsoft's annual company meeting at Safeco Field in Seattle when tens of thousands of highly connected employees gathered together in a single place.
Measures should (and are already being) implemented to ensure the resilience of data services to cope with the increasing demand from Smartphones. Governments and operators should continue to work together to explore available options, such as quality of service (QoS) provisions for emergency service levels of mobile data. For further reference, see the ENISA report Gaps in standardization related to resilience of communication networks.

2.5.1.3) Methods of security protection

Smartphones continue to grow in popularity and are now as powerful and functional as many computers. It is important to protect your Smartphone just like you protect your computer as mobile cyber-security threats are growing. Mobile security tips can help you reduce the risk of exposure to mobile security threats.

- **PINs and Passwords.**

To prevent unauthorized access to our phone, we set a password or Personal Identification Number (PIN) on phone's home screen as a first line of defense in case our phone is lost or stolen. When possible, we use a different password for each of our important log-ins (email, banking, personal sites, etc.). We should configure the phone to automatically lock after five minutes or less when the phone is idle, as well as we use the SIM password capability available on most Smartphones.

Recent research has shown that 54% of Smartphone users in the US do not set up password security on mobile phones – either when turned on or woken from standby. The reasons for doing so are obvious – if a phone is lost, stolen or simply left unattended, anyone that picks it up will have unrestricted access. This could involve data being stolen, phone calls being made or unwanted services being registered for, and could result in considerable financial cost.

There are a number of ways to protect a Smartphone. Many new phones offer a "pattern lock" – a personalized shape or pattern that is drawn on the screen to grant access, and this is often faster and less hassle than entering a password. Alternatively a PIN code offers a numeric alternative to a standard password and can also save time. Obviously a password that is easy to guess is less secure – so avoid "1234", "password" and other common phrases.

A screen lock is useful but won't stop someone from removing your SIM card and using it on another phone. To prevent this from happening, set up a SIM card lock in the form of a PIN number that will need to be entered when a phone is turned on in order to connect to a network.

With both of these security measures in place, you can at least be safe in the knowledge that if a phone is stolen it will be of very little use to the average thief.

No modification on Smartphone's security settings (rooting).

One increasingly popular practice among Android users is "rooting" a phone. This essentially involves modifying the file system to allow users access to read-only files and parts of the operating system that the manufacturer or service provider don't want you to change. Some of the advantages of rooting a phone include the ability to change or remove read-only applications that you don't want to use, change the boot screen, back up the entire system, run specialized applications and install custom user interfaces and alternative versions of the OS. **Rooting is usually only done by "experts", who should therefore be aware of the potential dangers, but if someone offers to root a phone for you while citing the benefits, it's important to be aware of the security risks as well.**

Since rooting allows a user access to system-level resources**, it also opens these up for potential infection by malware**. Part of the reason why this critical data is inaccessible is to protect it from such threats, and **while you may benefit from more flexibility in the short term, writers of malicious code can also benefit from full access to your device if it becomes infected.** Applications that have requested root access could, for example**, record keystrokes** entered on an on-screen keyboard, **delete or copy data**, **make phone calls** to premium numbers or **install "pseudo" applications** that look like the real thing, but have ulterior motives in mind.

This may sound like scaremongering, but it just goes to show the importance of being aware of the potential dangers involved with modern Smartphones, particularly flexible, open-source platforms like Android.

Backup and security of data.

Discovering that a phone has been lost or stolen is bad enough, but even when discounting the potential damage that could be done by sensitive data getting into the wrong hands, **important documents, contacts, messages, appointments and other information could take a long time to replace**. Ensuring that **regular backups are made is therefore essential**, and there are a number of ways to go about it. Most modern phones now allow users to “synchronize” information with a computer or website for productivity or backup purposes. This can include e-mails and contacts with Microsoft Outlook, photos uploaded to online storage or proprietary software supplied by the phone manufacturer to simply backup key data in the event of loss.

Some modern **security suites designed for use on mobile devices also offer an automatic backup facility** to take the hassle out of doing this manually. There are also a range of services that allow you to **automatically backup specific data to an online resource**, taking the hassle out of having to connect a phone to a computer. Provided you have a sufficiently healthy data plan, or are connected to a wireless network, this is an excellent way to safeguard against loss.

Identification and protection of sensitive data

Risks: Unsafe sensitive data storage, attacks on decommissioned phones unintentional disclosure: Mobile devices (being mobile) have a higher risk of loss or theft. Adequate protection should be built in to minimize the loss of sensitive data on the device.

- In the design phase, classify data storage according to sensitivity and apply controls accordingly (e.g. passwords, personal data, location, error logs, etc.). Process, store and use data according to its classification. Validate the security of API calls applied to sensitive data.
- Store sensitive data on the server instead of the client-end device. This is based on the assumption that secure network connectivity is sufficiently available and that protection mechanisms available to server side storage are superior. The relative security of client versus server-side security also needs to be assessed on a case-by-case basis.
- When storing data on the device, use a file encryption API provided by the OS or other trusted source. Some platforms provide file encryption APIs which use a secret key protected by the device unlock code and delete-able on remote kill. If this is available, it should be used as it increases the security of the encryption without creating extra burden on the end-user. It also makes stored data safer in the case of loss or theft. However, it should be born in mind that even when protected by the device unlock key, if data is stored on the device, its security is dependent on the security of the device unlock code if remote deletion of the key is for any reason not possible.
- Do not store/cache sensitive data (including keys) unless they are encrypted and if possible stored in a tamper-proof area.
- Consider restricting access to sensitive data based on contextual information such as location.

- Do not store historical GPS/tracking or other sensitive information on the device beyond the period required by the application.
- Assume that shared storage is un-trusted information may easily leak in unexpected ways through any shared storage. In particular:
  - ✓ Be aware of caches and temporary storage as a possible leakage channel, when shared with other apps.
  - ✓ Be aware of public shared storage such as address book, media gallery and audio files as a possible leakage channel. For example storing images with location metadata in the media-gallery allows that information to be shared in unintended ways.
  - ✓ Do not store temp/cached data in a world readable directory.
- For sensitive personal data, deletion should be scheduled according to a maximum retention period, (to prevent e.g. data remaining in caches indefinitely).
- There is currently no standard secure deletion procedure for flash memory (unless wiping the entire medium/card). Therefore data encryption and secure key management are especially important.
- Consider the security of the whole data lifecycle in writing your application (collection over the wire, temporary storage, caching, backup, deletion etc)
- Apply the principle of minimal disclosure - only collect and disclose data which is required for business use of the application. Identify in the design phase what data is needed, its sensitivity and whether it is appropriate to collect, store and use each data type.
- Use non-persistent identifiers which are not shared with other apps wherever possible - e.g. do not use the device ID number as an identifier unless there is a good reason to do so. Apply the same data minimization principles to app sessions as to http sessions/cookies etc.
- Applications on managed devices should make use of remote wipe and kill switch APIs to remove sensitive information from the device in the event of theft or loss.
- Application developers may want to incorporate an application-specific "data kill switch" into their products, to allow the per-app deletion of their application's sensitive data when needed.

Figure 19 Personal Data

Sensitive data in protected transit

Risks: Network spoofing attacks, surveillance. The majority of Smartphones are capable of using multiple network mechanisms including Wi-Fi, provider network (3G, GSM, CDMA and others), Bluetooth etc. Sensitive data passing through insecure channels could be intercepted.

- Assume that the provider network layer is not secure. Modern network layer attacks can decrypt provider network encryption, and there is no guarantee that the Wi-Fi network will be appropriately encrypted.
- Applications should enforce the use of an end-to-end secure channel (such as SSL/TLS) when sending sensitive information over the wire/air (e.g. using Strict Transport Security - STS).This includes passing user credentials, or other authentication equivalents. This provides confidentiality and integrity protection.
- Use strong and well-known encryption algorithms (e.g. AES) and appropriate key lengths.
- Use certificates signed by trusted CA providers. Be very cautious in allowing self-signed certificates. Do not disable or ignore SSL chain validation.
- For sensitive data, to reduce the risk of man-in-middle attacks (like SSL proxy, SSL strip), a secure connection should only be established after verifying the identity of the remote end-point (server). This can be achieved by ensuring that SSL is only established with end-points having the trusted certificates in the key chain.
- The user interface should make it as easy as possible for the user to find out if a certificate is valid.
- SMS, MMS or notifications should not be used to send sensitive data to or from mobile end-points.

Secure handling of password credentials on the device

Risks: Spyware, surveillance, financial malware. A user's credentials, if stolen, not only provide unauthorized access to the mobile backend service, they also potentially compromise many other services and accounts used by the user. The risk is increased by the widespread of reuse of passwords across different services.

- Instead of passwords consider using longer term authorization tokens that can be securely stored on the device. Encrypt the tokens in transit (using SSL/TLS). Tokens can be issued by the backend service after verifying the user credentials initially. The tokens should be time bounded to the specific service as well as revocable (if possible server side), thereby minimizing the damage in loss scenarios.
- In case passwords need to be stored on the device, leverage the encryption and key-store mechanisms provided by the mobile OS to securely store passwords, password equivalents and authorization

tokens. Never store passwords in clear text. Do not store passwords or long term session IDs without appropriate hashing or encryption.

- Some devices and add-ons allow developers to use a Secure Element e.g. sometimes via an SD card module - the number of devices offering this functionality is likely to increase. Developers should make use of such capabilities to store keys, credentials and other sensitive data. The use of such secure elements gives a higher level of assurance with the standard encrypted SD card certified at FIPS 140-2 Level 3. Using the SD cards as a second factor of authentication though possible, isn't recommended, however, as it becomes a pseudo-inseparable part of the device once inserted and secured.
- Provide the ability for the mobile user to change passwords on the device.
- Passwords and credentials should only be included as part of regular backups in encrypted or hashed form.
- Smartphones offer the possibility of using visual passwords which allow users to memorize passwords with higher entropy. These should only be used however, if sufficient entropy can be ensured.
- Swipe-based visual passwords are vulnerable to smudge-attacks (using grease deposits on the touch screen to guess the password). Measures such as allowing repeated patterns should be introduced to foil smudge-attacks.
- Check the entropy of all passwords, including visual ones.
- Ensure passwords and keys are not visible in cache or logs.
- Do not store any passwords or secrets in the application binary. Do not use a generic shared secret for integration with the backend (like password embedded in code). Mobile application binaries can be easily downloaded and reverse engineered.

Figure 20 Password wallet

Attention on open Wi-Fi networks.

Most Smartphones now have the option of connecting to wireless networks in the office or at home, or a wireless hotspot on the move. Opting for wireless is often beneficial for increased speeds or to save on data usage costs, so it’s easy to see why many prefer it when available**. Any device that’s enabled to send data across the airwaves is a potential security concern,** but thankfully modern phones are well prepared to help you mitigate this risk.

The first thing to remember is to always switch off a wireless connection when it’s not in use. Apart from helping you save on battery power, it ensures that malicious parties can’t connect to a device without your knowledge. It’s also worth taking a browse through a phone’s network security settings as **it might be configured to automatically connect to a network when in range.**

Wireless hotspots and unknown networks are by far the biggest risk when it comes to utilizing this connectivity – assuming of course, that any more commonly accessed wireless router in the home or office is sufficiently protected by a pass code.

A (relatively) common threat that pervades unknown wireless networks and hotspots is called the “evil twin” attack. Here **a malicious party might be offering access to a wireless connection that looks very much like a legitimate hotspot from a large company**. If a user were to inadvertently connect to this “hotspot”, they may find requests for passwords, login details and other information that can then be recorded and used to access their accounts at a later stage. If a little care is taken **it’s usually not too difficult to spot these attempts**, and of course any requests for information that don’t seem entirely legitimate and typical should be ignored.

Finally, those who use phones to communicate in a corporate environment should consider the use of a VPN (Virtual Private Network) to set up a secured private network. This allows users to access specific sites and company resources on the move and significantly reduces the risk of potentially sensitive data being intercepted by malicious parties.

Bluetooth

Unlike wireless networking, Bluetooth isn’t seen as a potentially risky venture for most mobile users, and the relatively short-range (around 10m) at which it is accessible does mean that it’s inherently safer. Attacks do still happen however, and it’s important to be aware of the pitfalls of leaving this technology switched on when not in use. Hackers have found ways to remotely access a phone (provided they are within range) and use it to make calls, access data, listen in on conversations and browse the internet.

To prevent this from happening it’s a good idea to set default Bluetooth configuration to “non-discoverable” mode by default. This means that users around you who are searching for potential targets won’t see your device pop up on their list.

It goes without saying that any unknown requests that come through via a Bluetooth connection, such as a request to “pair” with a device or respond to a message from an unknown source should be ignored or declined. Bear in mind that the restrictive range of Bluetooth means that other users or devices must be within this radius in order to connect to your device, and as such busy places such as coffee shops, bars, trains and buses have traditionally been opportunist environments for the Bluetooth hacker.

Acceptance on updates and patches for Smartphone's software.

We should keep ours phone's operating system software up-to-date by enabling automatic updates or accepting updates when prompted from your service provider, operating system provider, device manufacturer, or application provider. By keeping the operating system current, you reduce the risk of exposure to cyber threats.

Installation applications only from trusted sources.

Recent press surrounding malware on the Android operating system has reinforced the need to be cautious when downloading applications, and to pay attention to the requirements this software demands upon install. It's far too easy to simply breeze over these pages in an effort to get the app up and running, but users should exercise caution to ensure that realistic demands are being made on access to various features of a phone, particularly if the software isn't well known. While the Android Market recently succumbed to a malware care[17] it's generally far safer to use these "official" channels to download applications, and any secured from alternative sources should be treated as a potential risk.

It's also important to exercise caution with respected applications such as popular web browsers, as it's often far too easy to simply accept qualification messages that pop up when you're online. Agreeing to save user details and passwords when logging into websites for future access may be convenient, but makes it very easy for those accessing an unprotected phone to do the same. This is particularly important when it comes to online banks and merchants, as these sites often have bank account details saved automatically under your username and would make it easy for others to make unwanted purchases or transactions.

In addition users should pay attention to any potential security warnings that may be displayed when viewing websites, particularly if accessing them through unknown wireless networks, and not just dismiss these without thought. Web pages that involve the entry of sensitive data such as a username, password or account details should always use encrypted protocols to protect this information. This can be confirmed by the presence of an "s" at the end of "http" at the start of a webpage URL (https://) or a visible padlock icon on the status bar of a browser to confirm that the connection is encrypted.

Fully understanding of application permissions.

We should be cautious about granting applications access to personal information on our phone or otherwise letting the application have access to perform functions on your phone. Make sure to also check the privacy settings for each app before installing.

Installation of security apps that enable remote location and wiping.

An important security feature widely available on Smartphones, either by default or as an app, is the ability to remotely locate and erase all of the data stored on the phone, even if the phone's GPS is off. In the case that we misplace the phone, some applications can activate a loud alarm, even if the phone is on silent. These apps can also help us locate and recover the phone when lost.

[17] http://www.bullguard.com/bullguard-security-center/security-articles/mobile-security---the-deal-with-apps-for-android-phones.aspx

Wipe data from the previous device before donation, reselling or recycling.

Most Smartphones contain personal data that we want to keep private when we dispose our old phone. To protect privacy, we should completely erase data off of the phone and reset it its initial factory settings. After having wiped the old device, we are relieved to donate, resell, recycle, or otherwise properly dispose of the device.

Report a stolen Smartphone.

The major wireless service providers, in coordination with the FCC, have established a stolen phone database. If our phone is stolen, firstly we should report the theft to our local law enforcement authorities and then register the stolen phone with our wireless provider. This will provide notice to all the major wireless service providers that the phone has been stolen and will allow for remote “bricking” of the phone so that it cannot be activated on any wireless network without our permission.

2.5.1.4) The future Of Smartphone security

Modern Smartphones usually offer some kind of security option to let you into the phone. The iPhone has a four-digit PIN-like passcode, Android phones have a swiping pattern, Windows Phone has a numeric or text-based password. That work is OK, but they are **pretty hackable**, and could well be improved by new technologies. Insidious and scary technologies.

Figure 21 Iris for Scanning

The Problem

A large percentage of users select from a few easy-to-remember numbers. A four-digit PIN uses has around 10,000 possibilities, but the four most commonly used PIN numbers are used **about 20 percent of the time**. If the person trying to hack your phone knows anything about you--your birthday, when you were married, your address--it makes it that much easier to guess our PIN.

A PIN is easy to forget, which is why people often pick patterns that are easy to remember, like 1-2-3-4, which we should not use. And knowledge of the password is its only security; the security of an alphanumeric password does not involve the identity of the person entering it. The device does not attempt to see *who* is entering the password; if it's right, that's all that matters.

Figure 22 A Fingerprint

Biometrics: The Solution?

Biometrics refers to the practice of identifying someone based on physical characteristics, rather than on an alphanumeric key. The most well-known biometric technique is fingerprinting, but biometrics extends to all kinds of other ideas, from retinal scanning to analysis of a person's gait to detection of a person's particular smell (really). Some of these are good fits for Smartphone security! And some are not. (We probably won't be holding up the phone to our armpit for smell identification anytime soon.)

Fingerprinting: The grandpa of biometrics has a lot going for it. It requires minimal hardware, it's well-understood, and it's non-invasive. There are a few different ways to perform a fingerprint scan in a way that might be useful on a Smartphone. There's optical scanning, which is basically a digital camera; capacitance scanning, which measures the minute differences between electrical stimuli in the ridges and valleys of your fingerprint; and 3-D scanning, which makes a digital map of your finger, sort of like a Microsoft Kinect.

Capacitance scanning seems the most likely here; a company called AuthenTec has made some strides in capacitance fingerprint scanning on phones, partnering with Toshiba for the Regza T-01D Android phone. And, not at all coincidentally, Apple bought AuthenTec just over a year ago.

Will Apple opt for this method? Well, we haven't ever seen a scanner of this type integrated into a button. The iPhone has very few buttons; the home button sees a lot of action, and the last thing you want a fingerprint scanner to have to deal with is a lot of gunk and debris piling up in there from undue use. That's why the Toshiba Regza put the scanner on the back, where it's less likely to accumulate dust and dirt that could foul up its inner workings. And fingerprint quality is often degraded over time, thanks to repeated manual labor or skin buildup.

Face Recognition: Face recognition already exists! Ever since Android 4.0, back in 2011, Android phones have had face recognition as an unlocking option. The iPhone doesn't have face recognition built in, but we can snag an app that'll do it just as easily. Face recognition is great because it doesn't require any extra hardware--the front-facing camera on your phone is just fine for this--and, theoretically, when you turn on your phone, you've already got your face turned toward it, so you don't have to do much adjusting.

There are problems with this system because it's not particularly secure, for one thing. Like most biometrics that relies on mere optics, face recognition tools can be fooled with a good picture. It also depends on the quality of the teeny-tiny front-facing camera in the phone; what if it's dark? What if the camera breaks? What if there's blinding light? Any of those issues could make face recognition difficult.

Retinal Scanning: The retina is the part of the eye way at the back, a thin layer of tissue that serves basically the same function as a piece of film in a camera. It requires a complex series of blood vessels to feed it, and that configuration of blood vessels is unique to each person.

A retinal scan blasts some low-energy infrared light into the eye. The blood vessels don't reflect as much light as the surrounding tissue, so if we capture the image of the infrared reflection, we have got a unique map of the retina. It's *very* accurate, basically impossible to fool, and delivers results pretty much instantly. It's hard to fool with a fake retina; retinal scans can adjust to suit the movement of the retina upon getting blasted with the infrared light, so it's not easy to fool with a handheld fake eyeball.

Certain diseases, like glaucoma, can alter the pattern of blood vessels in the retina, as can eye conditions like astigmatism and cataracts. But those are minor problems. The real issue with retinal scanning is that it requires an infrared beam directed into your eyeball, which is invasive and not terribly convenient. Imagine having to stick your eye onto a scanner on the back of our phone every time we wanted to check your email.

Figure 23 Scanning Eyes

Iris Scanning: The iris is the colored part of the front of your eye, surrounding the pupil. The iris is also unique to each person, and much less likely to change over time than a fingerprint. It's also easy to scan, requiring only a camera and perhaps a slight infrared illumination to make things easier. The tech **already exists** as an external case, but wouldn't be too difficult to implement into a phone itself.

Iris scanning, though, can also **be fooled by a photo**. But it's harder to take a super sharp, high-definition photo of an iris compared to a person's face, which makes fooling an iris scanner with a photo relatively difficult. Still, it's far from impossible.

Electrocardiogram: According to a biometrics company called **Bionym**, your heartbeat is just as unique as your iris or fingerprint. The company very recently showed off its Nymi bracelet, which takes an electrocardiogram reading of your heart rate through your wrist. Then it uses Bluetooth to tell your phone to unlock.

It's a super promising technology; it could be nearly impossible to fool or replicate, low-power (because it relies on an external sensor), and cheap to implement.

What this Means?

Biometrics is almost certainly coming, in some form, to Smartphones. Whether it's this generation or an upcoming one, whether it's Apple or Samsung or Motorola, someone is going to figure out how to implement biometrics in a way that makes sense. Face recognition is already here!

But given the uproar over PRISM, people may also find it objectionable to subject themselves to any sort of identification, even if it's voluntary and even if it may assist with security. If we scan our retina to get into the iPhone, Apple has that data, Verizon (or whatever carrier we use) has that data, and we know from experience that if one of those companies has data, the government has access to it, too. And if the government has access to biometric data, that means they'll be able to link our fingerprints, heartbeat, iris, or retina to our email address, Twitter account, Facebook, YouTube and our Instagram. It would be very easy to create a huge database that properly associates physical identifiers to our digital life. And that's pretty scary.

Is biometrics a good way to secure a phone? Sure, if we are talking about merely stopping people from getting into our data when they physically have possession of the phone. But that's not really how hacking is done these days. If that happens, both Android and iOS allow us to remotely wipe our phone of all its data. Biometrics could have the totally unintended consequence of making us much *more* vulnerable to hacking, simply by making your physical phone more secure.

3) Android

3.1) Android Operation System (OS)

Overview

“The term “Android” has its origin in the Greek word andr-, meaning “man or male” and the suffix -eides, used to mean “alike or of the species”. This together means as much as “being human” ” (Speckmann, 2008).

According to Pew Research center, one of America’s think tank organization reports number of desktop owners declined and people are depending more on mobile phones and tablets (Janna Anderson, 2010). Today’s Smartphones are evolved from the conventional wired telephone system. Apple’s first smart phone iPhone became one of the best ever designed smart phone with its ease of use, portability and great computing power that no other company couldn’t make it. Apple iPhone’s operating system IOS is proprietary and Apple has got great control on the devices whoever uses it. To break this code Google acquired a small company called Android who is involved in the developing of mobile operating system. Google along with leading companies under the umbrella of Open Handset Alliance (OHA) started to develop an open source Linux based operating system. Google launched the first mobile phone called G1 based on Android after a year. From there onwards Google released several new versions of Android with new features and improvements.

There are several version of Android in the market and the latest Android version is version 5.0 code named as **Lollipop**. Android already made its mark being the fastest growing mobile platform. Gartner reported the worldwide smart phone sale with Android toping the market with 50% market share way ahead of all the other mobile platforms (Gartner, 2011). This is a clear indication of how Android based devices are capturing the markets of other mobile platforms.

Figure 24 Android Mobile OS 5.0 - Lollipop

What is really the Android Operation System

The Android OS is a Linux based open source operating system developed and maintained by Google. Android was designed to be executed on portable devices, such as smartphones and tablets. It provides a free and publicly available Software Development Kit (SDK) that consists of tools, documentation and emulators necessary for the development of new applications in Java.

A core element of the Android security model (Google, 2011c) is the manifest file. The manifest file is bundled into the Android installation package file (.apk file), along with the applications bytecode and other related resources. The file follows the XML structure and provides all the necessary information to the Android platform for the execution of the application. Security-wise the manifest file is crucial for the system, since a developer defines in it the permissions of each application. These permissions control: a) the way the application interacts with the system via access to system API, and b) the way the system and the other applications interact with the given application's components. By default, every application runs in a sandboxed environment without any permission to perform an action that can impact the operating system itself, the other applications and the user.

Every application requests authorization for its permissions at installation time, which is based on the user's approval. No further checks are made during the applications' execution. Hence, if the user decides to grant permission to the application, then the protected system resources are available to the application, otherwise the access to the resources is blocked

The installation package files of every Android application have to be digitally signed by its developer. Android's security model then maps the signature of the developer with a unique ID of the application package and enforces signature level permission authorization (Google, 2011c). However, in the Android security model it is not obligatory that the developer's certificate is signed by a trusted certificate authority. Thus, the applications are usually digitally signed with self-signed certificates, providing only poor source origin and integrity protection. This preserves the anonymity of a potential attacker, since the certificate is not verified by a Trusted Third Party (TTP).

A developer distributes her application either in the official Android application repository maintained by Google, the Android Market, or outside the repository. Google does not enforce any restriction-in the installation of applications originating outside its repository. On the other hand, Google developed technologies to remove applications (Google, 2011a) from the devices and the Android Market in case they pose a threat to the Android platform. The applications in the Android Market are provided to end users without being tested for malicious behavior. Hence, a developer must only provide her Google account and pay a small fee for the distribution of any application in the Android application repository. It is evident that this procedure is not able to stop malware from being distributed via the Android Market and being installed on devices.

History

Android, Inc. was founded in Palo Alto, California in October 2003 by Andy Rubin (co-founder of Danger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development at WebTV) to develop, in Rubin's words "smarter mobile devices that are more aware of its owner's location and preferences". The early intentions of the company were to develop an advanced operating system for digital cameras, when it was realised that the market for the devices was not large enough, and diverted their efforts to producing a smartphone operating system to rival those of Symbian and Windows Mobile (Apple's iPhone had not been released at the time). Despite the past accomplishments of the founders and early employees, Android Inc. operated secretly, revealing only that it was working on software for mobile phones. That same year, Rubin ran out of money. Steve Perlman, a close friend of Rubin, brought him $10,000 in cash in an envelope and refused a stake in the company.

Google acquired Android Inc. on August 17, 2005, making it a wholly owned subsidiary of Google. Key employees of Android Inc., including Rubin, Miner and White, stayed at the company after the acquisition. Not much was known about Android Inc. at the time, but many assumed that Google was planning to enter the mobile phone market with this move. At Google, the team led by Rubin developed a mobile device platform powered by the Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradable system. Google had lined up a series of hardware component and software partners and signaled to carriers that it was open to various degrees of cooperation on their part.

Speculation about Google's intention to enter the mobile communications market continued to build through December 2006. Reports from the BBC and the *Wall Street Journal*noted that Google wanted its search and applications on mobile phones and it was working hard to deliver that. Print and online media outlets soon reported rumors that Google was developing a Google-branded handset. Some speculated that as Google was defining technical specifications, it was showing prototypes to cell phone manufacturers and network operators. In September 2007, *InformationWeek* covered an Evalueserve study reporting that Google had filed several patent applications in the area of mobile telephony.

On November 5, 2007, the Open Handset Alliance, a consortium of technology companies including Google, device manufacturers such as HTC, Sony and Samsung, wireless carriers such as Sprint Nextel and T-Mobile, and chipset makers such as Qualcomm and Texas Instruments, unveiled itself, with a goal to develop open standards for mobile devices. That day, Android was unveiled as its first product, a mobile device platform built on the Linux kernel version 2.6. The first commercially available phone to run Android was the HTC Dream, released on October 22, 2008.

In 2010, Google launched its Nexus series of devices a line of Smartphones and tablets running the Android operating system, and built by a manufacturer partner. HTC collaborated with Google to release the first Nexus Smartphone, the Nexus One. The series has since been updated with newer devices, such as the Nexus 4 phone and Nexus 10 tablet, made by LG and Samsung respectively. Google releases the Nexus phones and tablets to act as their flagship Android devices, demonstrating Android's latest software and hardware features. On March 13, 2013, it was announced by Larry Page in a blog post that Andy Rubin had moved from the Android division to take on new projects at Google. He was replaced by Sundar Pichai, who also continues his role as the head of Google's Chrome division, which develops Chrome OS.

Since 2008, Android has seen numerous updates which have incrementally improved the operating system, adding new features and fixing bugs in previous releases. Each major release is named in alphabetical order after a dessert or sugary treat; for example, version 1.5 *Cupcake* was followed by 1.6 *Donut*. The latest released is **Android 5.0 "Lollipop"** which was unveiled on June 25, 2014 during Google I/O, it became available through official over-the-air (OTA) and it was updated on November 12, 2014, for select devices that run distributions of Android serviced by Google.

Features

Interface

Android's user interface is based on direct manipulation, using touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching and reverse pinching to manipulate on-screen objects. The response to user input is designed to be immediate and provides a fluid touch interface, often using the vibration capabilities of the device to provide haptic feedback to the user. Internal hardware such as accelerometers, gyroscopes and proximity sensors are used by some applications to respond to additional user actions, for example adjusting the screen from portrait to landscape depending on how the device is oriented, or allowing the user to steer a vehicle in a racing game by rotating the device, simulating control of a steering wheel.

Android devices boot to the home screen, the primary navigation and information point on the device, which is similar to the desktop found on PCs. Android home screens are typically made up of app icons and widgets; app icons launch the associated app, whereas widgets display live, auto-updating content such as the weather forecast, the user's email inbox, or a news ticker directly on the home screen. A home screen may be made up of several pages that the user can swipe back and forth between, though Android's home screen interface is heavily customizable, allowing the user to adjust the look and feel of the device to their tastes. Third-party apps available on Google Play and other app stores can extensively re-theme the home screen, and even mimic the look of other operating systems, such as Windows Phone. Most manufacturers, and some wireless carriers, customize the look and feel of their Android devices to differentiate themselves from their competitors.

Present along the top of the screen is a status bar, showing information about the device and its connectivity. This status bar can be "pulled" down to reveal a notification screen where apps display important information or updates, such as a newly received email or SMS text, in a way that does not immediately interrupt or inconvenience the user. In early versions of Android these notifications could be tapped to open the relevant app, but recent updates have provided enhanced functionality, such as the ability to call a number back directly from the missed call notification without having to open the dialer app first. Notifications are persistent until read or dismissed by the user.

Applications

Android has a growing selection of third party applications, which can be acquired by users either through an app store such as Google Play or the Amazon Apple Store, or by downloading and installing the application's APK file from a third-party site. The Play Store application allows users to browse, download and update apps published by Google and third-party developers, and is pre-installed on devices that comply with Google's compatibility requirements. The app filters the list of available applications to those that are compatible with the user's device, and developers may restrict their applications to particular carriers or countries for business reasons. Purchases of unwanted applications can be refunded within 15 minutes of the time of download, and some carriers offer direct carrier billing for Google Play application purchases, where the cost of the application is added to the user's monthly bill. As of September 2012, there were more than 675,000 apps available for Android, and the estimated number of applications downloaded from the Play Store was 25 billion.

Applications are developed in the Java language using the Android software development kit (SDK). The SDK includes a comprehensive set of development tools, including a debugger, software libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) plugin. Other development tools are available, including a Native Development Kit for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers, and various cross platform mobile web applications frameworks.

In order to work around limitations on reaching Google services due to Internet censorship in the People's Republic of China, Android devices sold in the PRC are generally customized to use state approved services instead.

Memory management

Since Android devices are usually battery-powered, Android is designed to manage memory (RAM) to keep power consumption at a minimum, in contrast to desktop operating systems which generally assume they are connected to unlimited mains electricity. When an Android app is no longer in use, the system will automatically suspend it in memory – while the app is still technically "open," suspended apps consume no resources (e.g. battery power or processing power) and sit idly in the background until needed again. This has the dual benefit of increasing the general responsiveness of Android devices, since apps don't need to be closed and reopened from scratch each time, but also ensuring background apps don't consume power needlessly.

Android manages the apps stored in memory automatically: when memory is low, the system will begin killing apps and processes that have been inactive for a while, in reverse order since they were last used (i.e. oldest first). This process is designed to be invisible to the user, such that users do not need to manage memory or the killing of apps themselves. However, confusion over Android memory management has resulted in third-party task killers becoming popular on the Google Play store; these third-party task killers are generally regarded as doing more harm than good.

Security and privacy

Android applications run in a sandbox, an isolated area of the system that does not have access to the rest of the system's resources, unless access permissions are explicitly granted by the user when the application is installed. Before installing an application, the Play Store displays all required permissions: a game may need to enable vibration or save data to an SD card, for example, but should not need to read

SMS messages or access the phonebook. After reviewing these permissions, the user can choose to accept or refuse them, installing the application only if they accept. The sandboxing and permissions system lessens the impact of vulnerabilities and bugs in applications, but developer confusion and limited documentation has resulted in applications routinely requesting unnecessary permissions, reducing its effectiveness. Several security firms, such as Lookout Mobile Security, AVG Technologies and McAfee, have released antivirus software for Android devices. This software is ineffective as sandboxing also applies to such applications, limiting their ability to scan the deeper system for threats.

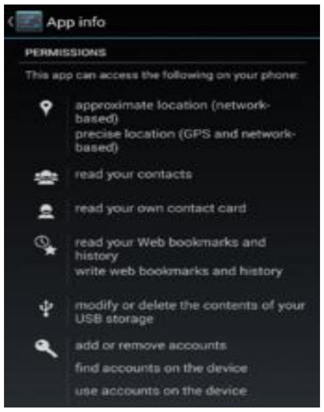
Research from security company Trend Micro lists premium service abuses as the most common type of Android malware, where text messages are sent from infected phones to premium-rate telephone numbers without the consent or even knowledge of the user. Other malware displays unwanted and intrusive adverts on the device, or sends personal information to unauthorized third parties.[121] Security threats on Android are reportedly growing exponentially; however, Google engineers have argued that the malware and virus threat on Android is being exaggerated by security companies for commercial reasons, and have accused the security industry of playing on fears to sell virus protection software to users. Google maintains that dangerous malware is actually extremely rare, and a survey conducted by F-Secure showed that only 0.5% of Android malware reported had come from the Google Play store.

Google currently uses their Google Bouncer malware scanner to watch over and scan the Google Play store apps. It is intended to flag up suspicious apps and warn users of any potential issues with an application before they download it. Android version 4.2 *Jelly Bean* was released in 2012 with enhanced security features, including a malware scanner built into the system, which works in combination with Google Play but can scan apps installed from third party sources as well, and an alert system which notifies the user when an app tries to send a premium-rate text message, blocking the message unless the user explicitly authorizes it.

Android Smartphones have the ability to report the location of Wi-Fi access points, encountered as phone users move around, to build databases containing the physical locations of hundreds of millions of such access points. These databases form electronic maps to locate Smartphones, allowing them to run apps like Foursquare, Google Latitude, Facebook Places, and to deliver location-based ads. Third party monitoring software such as TaintDroid, an academic research-funded project, can, in some cases, detect when personal information is being sent from applications to remote servers. In August 2013, Google released the Android Device Manager, a component that allows users to remotely track, locate, and wipe their Android device through an online interface. As it is implemented through Google Play Services instead of within Android itself, it is available to most Android devices with version 2.2 and higher.

The open-source nature of Android allows security contractors to take existing devices and adapt them for highly secure uses. For example Samsung has worked with General Dynamics through their Open Kernel Labs acquisition to rebuild *Jelly Bean* on top of their hardened micro-visor for the "Knox" project.

As part of the broader 2013 mass surveillance disclosures it was revealed in September 2013 that the American and British intelligence agencies, the NSA and Government Communications Headquarters (GCHQ) respectively, have access to the user data in iPhones, Blackberries, and Android phones. They are able to read almost all Smartphone information, including SMS, location, emails, and notes.


App info

PERMISSIONS

This app can access the following on your phone:

approximate location (network-based)
precise location (GPS and network-based)

read your contacts

read your own contact card

read your Web bookmarks and history
write web bookmarks and history

modify or delete the contents of your USB storage

add or remove accounts
find accounts on the device
use accounts on the device

Figure 25 App info from Android

3.2) Android Background

3.2.1) Partitions

Like in traditional computers, the disk space on an Android device can contain be divided into multiple partitions for better organization. Typically, any Android device will contain six important partitions:

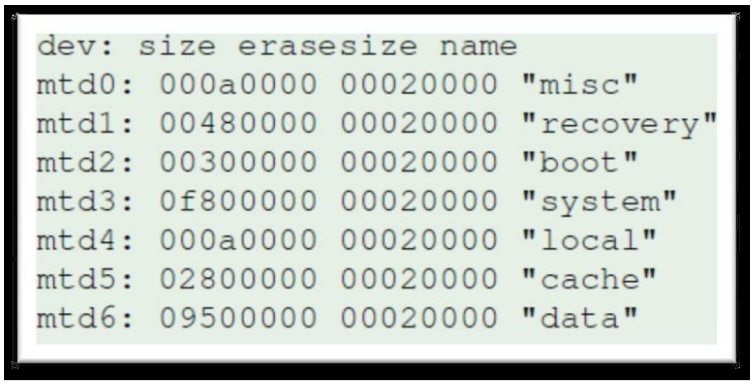

```
dev:  size  erasesize name
mtd0: 000a0000 00020000 "misc"
mtd1: 00480000 00020000 "recovery"
mtd2: 00300000 00020000 "boot"
mtd3: 0f800000 00020000 "system"
mtd4: 000a0000 00020000 "local"
mtd5: 02800000 00020000 "cache"
mtd6: 09500000 00020000 "data"
```

Figure 26 Android Partitions

- **Misc**: Contains miscellaneous system settings such as carrier or region id, USB configuration, and certain hardware settings. The device will not operate properly without this partition.

- **Recovery**: A very important partition that acts as an alternate boot partition that can be used to perform advanced recovery or maintenance operations.

- **Boot**: This enables the phone to boot, and includes the kernel and ramdisk. If this partition is corrupted or missing the phone will not be able to boot at all.

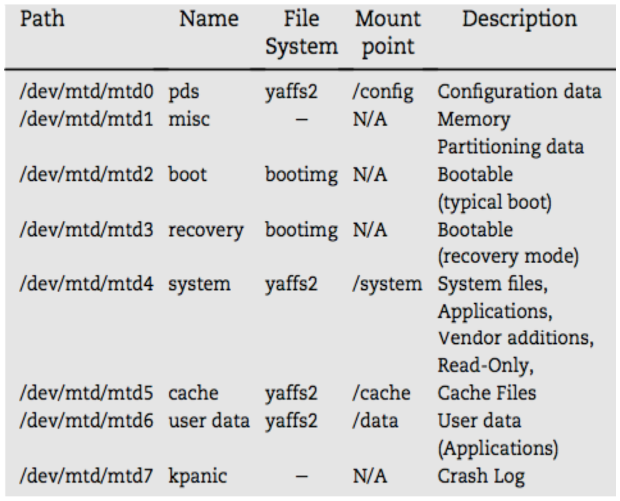
- **System**: The entire operating system is contained here (other than the kernel and ramdisk), including the Android interface and the preinstalled applications that come on the phone.

- **Cache**: This partition (as the name suggest) is used by the OS to cache various information such as frequently accessed data and application components.

- **Data**: User's data such as contacts, messages, settings, and user installed application are contained in this partition. Erasing this partition essentially restores the device back to its initial factory state, making the device boot how it did the first time a user powered it on.

3.2.2) Android File system

A file system is way to organize the data in an efficient manner. There are different file system for computers and mobile phones. The efficiency of a file system is weighed on how fast the application can read, write and retrieve data. Android is based on Linux file system and many of them used to boot and run the device. Android uses EXT, FAT32 and YAFFS2 file systems for booting, data storage purposes. Extraction tools use many methods to pull the data out of these file systems. FAT and FAT32 are popular file system used by windows operating system. Android supports these files system mainly in the SD Cards. Although these files system lacks the security of the files this is used widely. YAFFS2 (Yet another Flash File System 2) a file system designed for flash memory. The problem with YAFFS is that most forensic tools available are not compatible with this file system. But researcher Andrew Hoog pointed out that some Android handsets were already using EXT4. This is due to the support for the dual core processor and multiprocessing and to use external memory cards (Hoog, Android Forensics, 2011).
Vidas, et al., (2011) describes different partitioning used in Android devices but exact partitioning depends upon handset manufacturer. The typical scheme found in the Android device is shown in the Figure 2.2. There are normally six partitions found on Android devices, user data, cache, boot, and recovery. The YAFFS2 file system was designed for flash memory. Vidas, et al., outlines methods of collection by using recovery partition. This piece of information is very critical while doing the forensic analysis of a mobile device.

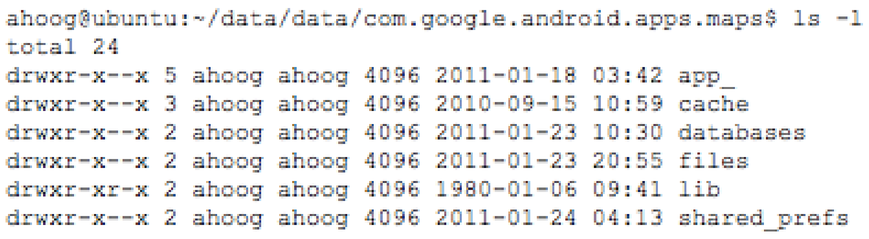


Path	Name	File System	Mount point	Description
/dev/mtd/mtd0	pds	yaffs2	/config	Configuration data
/dev/mtd/mtd1	misc	–	N/A	Memory Partitioning data
/dev/mtd/mtd2	boot	bootimg	N/A	Bootable (typical boot)
/dev/mtd/mtd3	recovery	bootimg	N/A	Bootable (recovery mode)
/dev/mtd/mtd4	system	yaffs2	/system	System files, Applications, Vendor additions, Read-Only,
/dev/mtd/mtd5	cache	yaffs2	/cache	Cache Files
/dev/mtd/mtd6	user data	yaffs2	/data	User data (Applications)
/dev/mtd/mtd7	kpanic	–	N/A	Crash Log

Figure 27 Partition Information of an Android device (Timothy Vidas, 2011)

3.2.3) Android Data storage

Android phones store more data than any standard phone. Android stores data in five methods. Forensic examiners usually look for the data in four of the five formats.
The five methods of storing data are shared preferences, internal storage, external storage, SQL lite and network (Hoog, Android Forensics, 2011). Shared preferences allow a developer to store key-value pairs of primitive data types in a lightweight XML format. Shared preferences files are stored in an application's data directory in the 'shared_pref' folder.
Android phones store much of the data in internal memory such as NAND flash. The files are stored in the applications /data/data folder. This folder can only visible if a root (administrator) access has gained.



```
ahoog@ubuntu:~/data/data/com.google.android.apps.maps$ ls -l
total 24
drwxr-x--x 5 ahoog ahoog 4096 2011-01-18 03:42 app_
drwxr-x--x 3 ahoog ahoog 4096 2010-09-15 10:59 cache
drwxr-x--x 2 ahoog ahoog 4096 2011-01-23 10:30 databases
drwxr-x--x 2 ahoog ahoog 4096 2011-01-23 20:55 files
drwxr-xr-x 2 ahoog ahoog 4096 1980-01-06 09:41 lib
drwxr-x--x 2 ahoog ahoog 4096 2011-01-24 04:13 shared_prefs
```

Figure 28 "shared_pref" folders under Application data

SD Cards are normally used as external storage and loaded with FAT32 file system. FAT32 is widely supported but lacks any security mechanism like in ext3, ext4, yaffs2 etc. SQLite is a popular database format appearing in many mobile systems.
SQLite is very lightweight and its entire code base is of high quality and open source. The SQLite files are generally stored on the internal storage under /data/data/<packageName>/database. Network can be used to store and retrieve data.

3.2.4) Boot Process

Understanding the Android boot process is essential in knowing how the operating system functions and what is performed at boot time. Looking into the source code we can see very interesting things in the startup scripts of what is being done. There are six fundamental steps in Android boot process which I'll briefly go over:

- **Power on and on-chip boot ROM code execution**: This step is hardware specific in how the processor and other hardware units initialize on power-up, but the important part is that this is where the boot media is located and the bootloader is then copied to internal RAM and then executed.

- **The Boot Loader**: Here the Initial Program Loader (IPL) detects and sets up the external RAM, copies the Second Program Loader (SPL) to internal RAM and transfers execution to it. The SPL is in charge of loading the OS and other possible mode such as the Recovery partition or the Fastboot protocol. SPL copies the Linux kernel that is located on the boot media to RAM and then transfers execution to the kernel.

- **The Linux Kernel**: Now that the Linux kernel is loaded it begins its usual boot process including setting up more features.

- **The init process**: This is similar to the same init.d process that computer running Linux goes through when booting up. Here startup scripts are ran to setup and do various functions, and start core services.

- **Zygote and Dalvik**: This next process is called the Zygote process and is in charge of setting up the Java runtime environment (Dalvik), and then registers a socket with the system so it can communicate. Now applications can run by requesting new Dalvik virtual machines that each one runs in.

- **The System Server**: Finally the system server begins running which takes care of core features like telephony, networking, and others.

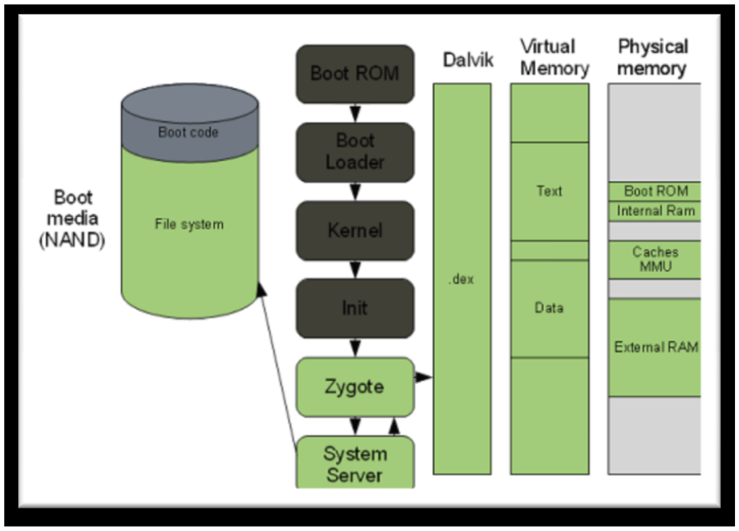


Figure 29 Android Boot Process

3.2.5) Boot loader

The Android bootloader is what is run before any part of the actual operating system is loaded. It can be accessed on any Android device by holding down a device specific combination of buttons on power-up. This is important because we want to be able to access the disk that the data is stored on while bypassing the filesystem so that a bit-by-bit extraction can be performed. Here is where we look to somehow load a program to be able to do this. Luckily the bootloader has more options other than just booting into the operating system. One option is to load the fastboot protocol which allows you to modify the various partitions on the device. Unfortunately, this option is not always present, and even if it is, is usually unable to modify partitions do to a security feature enabled by most manufacturers. The other option at the bootloader menu is to execute the Recovery partition, but it is simply preloaded onto the device by the manufacturer to recover or service the device if corruption occurs.

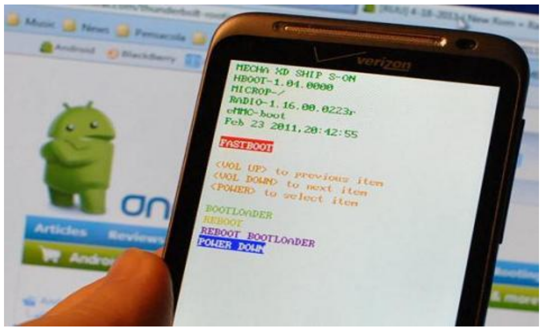
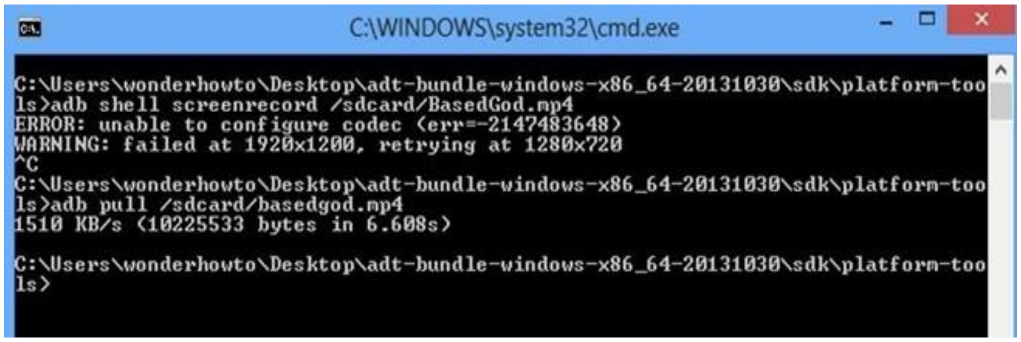

```
MECHA XD SHIP S-ON
HBOOT-1.04.0000
MICROP-/
RADIO-1.16.00.0223r
eMMC-boot
Feb 23 2011,20:42:55

FASTBOOT

<VOL UP> to previous item
<VOL DOWN> to next item
<POWER> to select item

BOOTLOADER
REBOOT
REBOOT BOOTLOADER
POWER DOWN
```

Figure 30 Boot Loader

3.2.6) Android Debug Bridge (ADB)

The Android Debug Bridge (ADB) is included with the Android Software Development Kit which is free to download and use. ADB is a command line tool that allows you to communicate with an Android device through usb to transfer files, run a remote shell, and some other commands. Any Android can be used with the ADB by enabling debugging mode in the settings menu located at Settings→Applications→Development→USB debugging. As mentioned before the ADB can be used to retrieve some files from the device but since this is through the filesystem, permissions are enforced so that not all files are accessible as well as the OOB information.



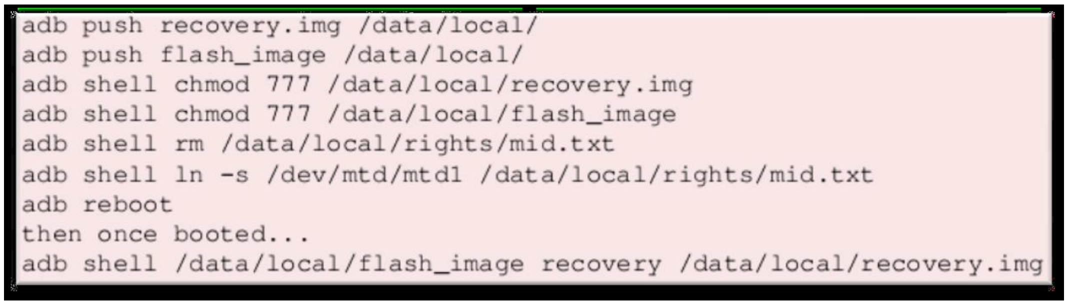
```
C:\Users\wonderhowto\Desktop\adt-bundle-windows-x86_64-20131030\sdk\platform-too
ls>adb shell screenrecord /sdcard/BasedGod.mp4
ERROR: unable to configure codec (err=-2147483648)
WARNING: failed at 1920x1200, retrying at 1280x720
^C
C:\Users\wonderhowto\Desktop\adt-bundle-windows-x86_64-20131030\sdk\platform-too
ls>adb pull /sdcard/basedgod.mp4
1510 KB/s (10225533 bytes in 6.608s)

C:\Users\wonderhowto\Desktop\adt-bundle-windows-x86_64-20131030\sdk\platform-too
ls>
```

Figure 31 Android Debug Bridge (ADB)

3.2.7) Recovery Partition

The Recovery partition is preloaded onto the device by the manufacturer in case the phone needs to be serviced or formatted from an irrecoverable state. The idea is to overwrite the Recovery partition with a custom image so that it can be ran from the bootloader, therefore giving us full access to the disk because the operating system and filesystem are never loaded. The problem is that since we can't use fastboot, the Recovery partition is only accessible through the filesystem by which it's is protected by.

However, an exploit was discovered in the Android operating system that allows us to modify this partition's permissions so that we are freely able to modify it. The exploit is within the startup scripts executed by the kernel during the boot process. One script in particular modifies the permission of a file so that is can be modified by any user. Therefore, using the ADB, we can simply replace the file being modified by the startup script with a symbolic link (named the same as the file) pointing to the recovery partition. Now, once the device reboots, the startup script runs and modifies the permission bits on the recovery partition (thinking it is the usual file), giving us full access to it. Now we can use ADB to put our custom recovery image and a simple program on the device, and use the program to flash our custom image to the recovery partition. The commands to do this are as listed here:

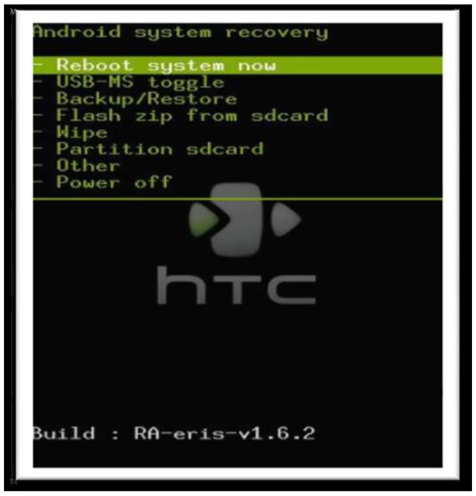


```
adb push recovery.img /data/local/
adb push flash_image /data/local/
adb shell chmod 777 /data/local/recovery.img
adb shell chmod 777 /data/local/flash_image
adb shell rm /data/local/rights/mid.txt
adb shell ln -s /dev/mtd/mtd1 /data/local/rights/mid.txt
adb reboot
then once booted...
adb shell /data/local/flash_image recovery /data/local/recovery.img
```

Figure 32 Recovery Partition

3.2.8) Custom Recovery ROM

The custom recovery image (“ROM’ and “image” can be used interchangeably) we flash onto the device recovery partition needs a few essential things to allow us to perform data extraction. First, a communication protocol needs to be employed so that we can transfer data to/from the device. For this, the ADB is from the Android SDK since it is open source. Next we need something to be able to execute commands and run the tools we need to use. Here, software called BusyBox is employed which is an essentially stripped-down Unix tool that are combined into a single executable (it can be thought of as a tiny version of linux). Lastly, we need a tool that can actually extract the data from the device. We could simply use dd for this, but I utilized a tool called NANDroid which is specifically written for use on Android devices with flash disks. NANDroid simply creates bit-by-bit exact image copies for any of the partitions on the device. Instead of building our own recovery image I used a popular one that already existed called “Amon_RA” which contains all of these utilities. Now that we have our custom image flashed in the recovery partition, when we boot up the device to the bootloader and select the recovery partition we are greeted with a custom menu that gives access to various functions that can be ran from the device itself, as well as ADB running in the background so that we may run a remote shell that gives us full access to the device.



```
Android system recovery

- Reboot system now
- USB-MS toggle
- Backup/Restore
- Flash zip from sdcard
- Wipe
- Partition sdcard
- Other
- Power off

Build : RA-eris-v1.6.2
```

Figure 33 Android System Recovery

3.3) The current state of Android

Initially fuelled by the introduction of the iPhone, the Smartphone market is growing exponentially, but there is one OS in particular that has been enjoying particularly rapid development recently. That system is Google's Android, which according to Gartner has grown from a 23% market share last year, to become the dominant market force at 38% in 20111. As the user base has grown so too has the risks associated with the OS. The reason for the rise of Android is choice – consumers are able to choose from a broad range of manufacturers and price levels opposed to Apple's mono-device approach.

Despite some challenging the impact of mobile malware as overhyped, new figures would suggest otherwise. Mobile malware was always rife on Symbian but rapid adoption of mobile platforms by both personal and business users has seen to become more attractive to cybercriminals. In the last year, 20% of all cybercrime in UAE occurred on mobile devices2 and Goode Intelligence found that up to 18% of organizations in the UK had experienced a mobile malware incident3. Such figures demonstrate that it is not just consumers who are at risk to such threats but enterprises too that are failing to implement effective mobile security measures. Trends in mobile threats have matched what we see at a wider industry level, with Android becoming the most-targeted of the mobile platforms, according to McAfee, much as Windows has on the PC. The number of Android devices under botnet control has peaked at 40,000 Android devices worldwide on several occasions4, and the problem is getting worse with Lookout claiming 0.5-1 million users were infected in the first half of 20115. In Q2 this year there were no new malware signatures detected for iOS, whereas there were 44 for Android in that same time period6. The threat is very real, and rapidly developing. In emerging as the top mobile platform, Google's little green Android has painted a big red target on itself.

The threats we see appearing on mobile, are not going to be a new concept to the security professional – Rootkits, Trojans, and even Botnets are making appearances. Indeed the SpyEye Botnet has successfully transitioned from the PC to Android, highlighting the commercialization of mobile malware. The Android incarnation of SpyEye, Spitmo, utilises a familiar approach, namely convincing users it is a legitimate security measure for their device. Mobile malware such as this strive not just to acquire data but also to monetize the attack by using the device to access premium SMS and voice services.

The open nature of Android and its large user base have made it an attractive and profitable platform to attack. Common exploits and tool kits on the OS can be utilized across a wide number of devices, meaning that attackers can perform exploits en masse and re-use attack vectors. It is obvious why Android is a target, but why is it vulnerable? Google did take measures in the development of the Android kernel to build security measures in; the OS is sandboxed, preventing malicious processes from crossing between applications. Whilst this attempt to eliminate the concept of infection is admirable in some regards, it fails to address the issue of infection altogether.

Android is a victim of its own success, not just in the way it has attracted malicious attention, but in its very nature. One of the reasons the OS has succeeded in gaining market share so rapidly is that it is open source is essentially free for manufacturers to implement. Additionally this has led to substantial fragmentation of Android versions between devices and means that vendors have been reluctant to roll-out updates, presumably out of some concern regarding driving demand for future devices. There is little value to the manufacturer in updating a device, something that to date Google has tried to encourage but been largely unsuccessful in doing so. Where updates do occur, manufacturer specific software on top of Android (such as HTC’s Sense or Motorola’s Blur) and even network provider bloatware, serve only to further delay patch management. After Google release an update this must then be customized by the manufacturer and network before release, unless of course it is a vanilla device such as the Nexus range. As a result vulnerabilities are left unpatched in stock ROMs, and advanced users are turning to flashing custom ROMs on their devices which raises a whole host of other issues.

Increasingly employees want to be able to use their Smartphones at work, they want to access their email on the go, may need to access a content management system, and might prefer to log on to the corporate network than use 3G. Where Blackberry went from enterprise to consumer in terms of market penetration, Android is doing the inverse, where consumers are buying these devices for personal use but wanting to utilize them in a professional capacity as well but without regard for the impact.

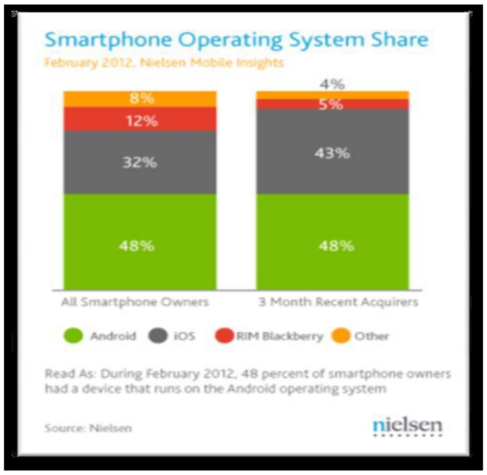


Figure 34 Percentages of Mobile Operation Systems

3.4) Android security

Overview

Android is a modern mobile platform that was designed to be truly open. Android applications make use of advanced hardware and software, as well as local and served data, exposed through the platform to bring innovation and value to consumers. To protect that value, the platform must offer an application environment that ensures the security of users, data, applications, the device, and the network.

Securing an open platform requires a robust security architecture and rigorous security programs. Android was designed with multi-layered security that provides the flexibility required for an open platform, while providing protection for all users of the platform.

Android was designed with developers in mind. Security controls were designed to reduce the burden on developers. Security-savvy developers can easily work with and rely on flexible security controls. Developers less familiar with security will be protected by safe defaults.

Android was designed with device users in mind. Users are provided visibility into how applications work, and control over those applications. This design includes the expectation that attackers would attempt to perform common attacks, such as social engineering attacks to convince device users to install malware, and attacks on third-party applications on Android. Android was designed to both reduce the probability of these attacks and greatly limit the impact of the attack in the event it was successful.

This document outlines the goals of the Android security program, describes the fundamentals of the Android security architecture, and answers the most pertinent questions for system architects and security analysts. This document focuses on the security features of Android's core platform and does not discuss security issues that are unique to specific applications, such as those related to the browser or SMS application. Recommended best practices for building Android devices, deploying Android devices, or developing applications for Android are not the goal of this document and are provided elsewhere.

Background

Android provides an open source platform and application environment for mobile devices.

The main Android platform building blocks are:

- **Device Hardware**: Android runs on a wide range of hardware configurations including smart phones, tablets, and set-top-boxes. Android is processor-agnostic, but it does take advantage of some hardware-specific security capabilities such as ARM v6 eXecute-Never.
- **Android Operating System**: The core operating system is built on top of the Linux kernel. All device resources, like camera functions, GPS data, Bluetooth functions, telephony functions, network connections, etc are accessed through the operating system.
- **Android Application Runtime**: Android applications are most often written in the Java programming language and run in the Dalvik virtual machine. However, many applications, including core Android services and applications are native applications or include native libraries. Both Dalvik and native applications run within the same security environment, contained within the Application Sandbox. Applications get a dedicated part of the filesystem in which they can write private data, including databases and raw files.

Android applications extend the core Android operating system. There are two primary sources for applications:

- **Pre-Installed Applications**: Android includes a set of pre-installed applications including phone, email, calendar, web browser, and contacts. These function both as user applications and to provide key device capabilities that can be accessed by other applications. Pre-installed applications may be part of the open source Android platform, or they may be developed by an OEM for a specific device.
- **User-Installed Applications**: Android provides an open development environment supporting any third-party application. Google Play offers users hundreds of thousands of applications.

Google provides a set of cloud-based services that are available to any compatible Android device. The primary services are:

- **Google Play**: Google Play is a collection of services that allow users to discover, install, and purchase applications from their Android device or the web. Google Play makes it easy for developers to reach Android users and potential customers. Google Play also provides community review, application license verification, application security scanning, and other security services.
- **Android Updates**: The Android update service delivers new capabilities and security updates to Android devices, including updates through the web or over the air (OTA).
- **Application Services**: Frameworks that allow Android applications to use cloud capabilities such as (backing up) application data and settings and cloud-to-device messaging (C2DM) for push messaging.

Android Security Program Overview

Early on in development, the core Android development team recognized that a robust security model was required to enable a vigorous ecosystem of applications and devices built on and around the Android platform and supported by cloud services. As a result, through its entire development lifecycle, Android has been subjected to a professional security program. The Android team has had the opportunity to observe how other mobile, desktop, and server platforms prevented and reacted to security issues and built a security program to address weak points observed in other offerings.

The key components of the Android Security Program include:

- **Design Review**: The Android security process begins early in the development lifecycle with the creation of a rich and configurable security model and design. Each major feature of the platform is reviewed by engineering and security resources, with appropriate security controls integrated into the architecture of the system.
- **Penetration Testing and Code Review**: During the development of the platform, Android-created and open-source components are subject to vigorous security reviews. These reviews are performed by the Android Security Team, Google’s Information Security Engineering team, and independent security consultants. The goal of these reviews is to identify weaknesses and possible vulnerabilities well before the platform is open-sourced, and to simulate the types of analysis that will be performed by external security experts upon release.
- **Open Source and Community Review**: The Android Open Source Project enables broad security review by any interested party. Android also uses open source technologies that have undergone significant external security review, such as the Linux kernel. Google Play provides a forum for users and companies to provide information about specific applications directly to users.
- **Incident Response**: Even with all of these precautions, security issues may occur after shipping, which is why the Android project has created a comprehensive security response process. A full-time Android security team constantly monitors Android-specific and the general security community for discussion of potential vulnerabilities. Upon the discovery of legitimate issues, the Android team has a response process that enables the rapid mitigation of vulnerabilities to ensure that potential risk to all Android users is minimized. These cloud-supported responses can include updating the Android platform (over-the-air updates), removing applications from Google Play, and removing applications from devices in the field.

3.4.1) Security architect

Without a doubt, security threats are escalating. As more people use mobile phones and tablets for high-value activities such as banking and payments, mobile devices become an increasingly lucrative target for criminals. Malware for Android, as for all mobile OSs, is rising in both volume and sophistication.

Android was designed to have rigorous, multilayered security with an expectation that attackers will use social engineering and malware to compromise the user and the device. The platform itself is designed to protect user data and system resources and to provide application isolation. Android provides security at the operating system level through the Linux kernel.

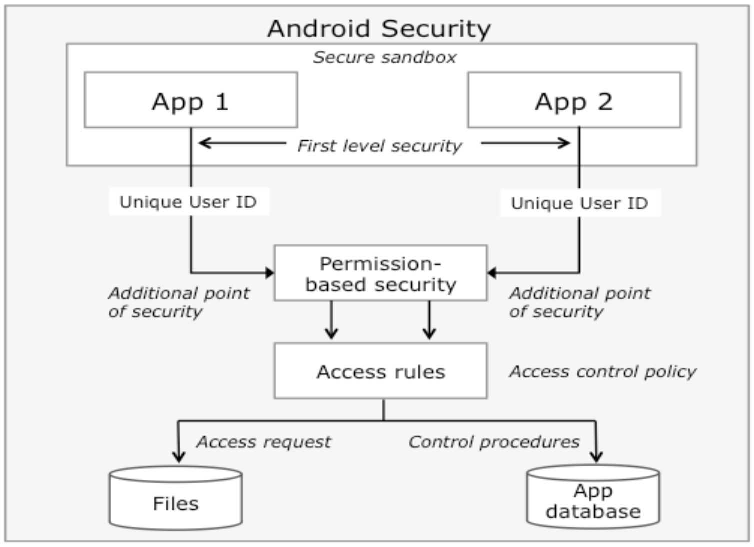


Figure 35 Android Security Architecture

With Android, each application runs a distinct system identity, and parts of the system are also separated into distinct identities. Android thereby isolates, or *sandboxes*, applications from each other and from the system. This means applications must share resources and data by declaring the permissions they require. The Android system prompts the user for consent at the time the application is installed (using the permissions model discussed earlier in the paper). Android has no method to grant permissions dynamically at run-time because it complicates the user experience to the detriment of security. That means an attacker must compromise the security of the Linux kernel to break out of an application sandbox.

Communications between components in the application or between applications themselves take place via *intents*. An intent is an abstract description of an operation to be performed or a description of something that has happened and is being announced. The Android system finds the appropriate activity, service or broadcast based on the Intent. Using Intents allows services and applications to communicate dynamically while maintaining the isolation provided by sandboxing.

For instance, if the user finds a restaurant on Yelp, and then wants to create a reservation, the Yelp application broadcasts reservation *intent*. Any application on the device, which can process that intent, is made available to use. In this scenario, the user would be given the option to create the reservation using the OpenTable application.

Android Platform Security Architecture

The beauty of Android design is its compatibility with wide range of hardware's. This is achieved mainly due to its Linux kernel which in fact famous for its compatibility to many different hardware platforms. This is the feature that allows the manufacturers freedom to design the devices according to their requirements. But the fact is that the flexibility of Android is big challenge for a forensic investigator. So the understanding of the Android internals and the architecture is at most important. Android platform is keep changing with the time. Though these changes effectively change the architecture also but there are core components that are static any time. The Android architecture (Figure) consists of a software stack, which combines an operating system, middleware and application framework. The software stack is divided in four different layers.

The basic of Android architecture is Linux kernel, which is open-source and widely studied, allowing us to be able to utilize preexisting knowledge and utilities. Android OS is also open-source itself which allows us to download its source code and analyze its operation down to the lowest level so that we are able to fully understand its operation.

Android also seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to:

- ✓ Protects user data
- ✓ Protects system resources (including the network)
- ✓ Provides application isolation

To achieve these objectives, Android provides these key security features:

- Robust security at the OS level through the Linux kernel
- Mandatory application sandbox for all applications
- Secure interprocess communication
- Application signing
- Application-defined and user-granted permissions

The sections below describe these and other security features of the Android platform. The picture of Figure 36 summarizes the security components and considerations of the various levels of the Android software stack. Each component assumes that the components below are properly secured. With the exception of a small amount of Android OS code running as root, all code above the Linux Kernel is restricted by the Application Sandbox.

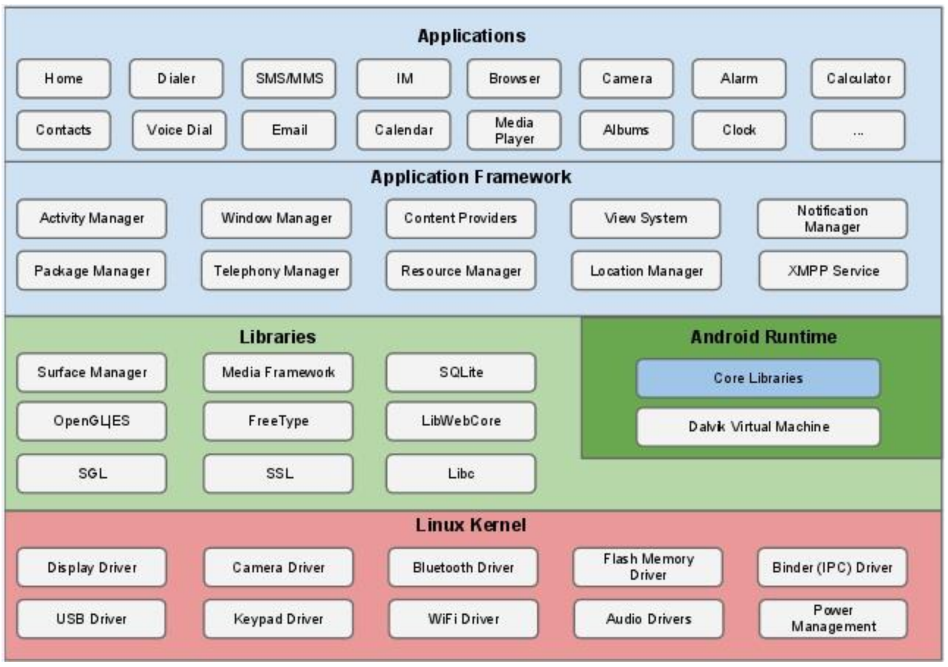


Figure 36 Android software Stack

Linux Kernel

Linux 2.6 kernel is the core of the Android platform. Linux kernel is the hardware abstraction layer that controls the hardware and its resources. The kernel also controls process management, memory management, network management and power management and proven to be very stable. Kernel maintains various drivers for almost all of the hardwares (Hoog, Android Forensics, 2011).

Libraries

The next layer in the Android Architecture is the Libraries. The libraries are written in C/C++. The core power of Android platform is utilized through the libraries like surface manager for composing different drawing surfaces onto the screen. The openGL/ES and SGL makes the core of the 3D and 2D graphics and the media framework constitute various media packs like MPEG, AAC, mp3, mp4 etc., SQL lite for the data storage, WebKit for the open source browser engine and freetype for the font engine (Sayed Hashimi, 2010).

Android Runtime

Android Runtime built mainly to support its embedded environment with limited CPU, battery and small screen. Dalvik machines run dex files that are byte code converted from jar files. Dalvik machines are customized version of Java suited in the small platform environment. Dalvik uses highly optimized CPU, data shared to other applications such a way that there might be multiple instances of Dalvik virtual machines at a given time. The core libraries are the all the java libraries for all the classes (Ehringer, 2010).

Application Framework

Application framework is written in Java. This is toolkit that all the applications uses are the application written by Google or the application written by the developers. All the applications use the same framework and same APIs making it simple to write applications. By looking all the components, the activity manager manages the application life cycle. The package manager keeps track of all the application presently in the device and updates if any new application been downloaded and installed. Window manager manages the windows UI.
Telephony manager uses APIs that used to build the telephone applications, content provider is unique that share the data across all the applications like contacts are shared to all the applications. Location manager and notification manager allow developers to develop new, exciting and innovative pieces of applications.

Applications

Android comes with set of basic application like Contacts, Calendar, maps, SMS app, Email client and browser. All of these applications are written in Java programming language. These applications are multi tasking. All applications run its own processes. Android provides a feature rich APIs for building innovative applications. APIs like Location manager and XMPP (Extensible Messaging and Presence Protocol) services gives developers ability to build applications. Notification manager APIs to build the notification related applications.

Android SDK & ADB

One of the important features available in the Android Software Development kit (SDK) is the Android Debug Bridge (ADB). ADB provides a communication interface to an Android system using a computer. A computer is able to access a command shell, install or remove applications, transfer files when connected through ADB interface (Android , 2012).
Access to system partitions is restricted to the Android operating system. By default, users do not have permission to access system reserved areas. This is mainly to prevent malicious or poorly developed applications to affect Android OS stability and reliability. But as a forensic point of view this may be a bad idea, but yes, it is possible to get super user or root access to the system. A forensic analyst always looking ways to get into the internals of the system so getting access to the root is very important. Forensic analyst can make a copy of all the system partitions as well as access files that are not accessible by normal access. But again this depends on the device manufacturer and model and also invasive in nature. The Android SDK not only provides deep insight into the Android platform but also provides powerful tools to investigate the device, from both a forensic and security viewpoint. Once the SDK is installed on a forensic workstation, the forensic examiner has the ability to interact with an Android device connected via USB, provided the USB debugging feature is enabled. The Android SDK is an important tool used for forensic and security analysis. For a comprehensive forensic analysis the forensic investigator required to know the boot process of Android device. So like any other device Android also has fairly standard boot process that allows the device to load the required firmware, OS and user data into memory to support full operation. Mattias Bjornheden of the Android Competence Center at Enea described in detail how and what are the methods of Android boot process in his blog post (Bjornheden, 2009).

In this post they are identified seven steps to Android boot process and they are:

- Power on and on-chip boot ROM code execution
- The boot loader
- The Linux kernel
- The init process
- Zygote and Dalvik
- The system server
- Boot complete

The device components vary from one manufacture to another but a basic understanding of these components and device type is helpful for the forensic analyst for a clear and concise investigation.
Android updates are not a centralized repository of updates where users run the updates from a central server. Instead the updates are the responsibilities of the carriers or the network service providers. Google take full responsibility of managing the Android OS as whole and not the updates. For a forensic analysts perspective this is a bit troublesome in many ways. Firstly a forensic investigator is never certain what version of Android a device is installed. There are a number of versions of Android in the market and they vary in their features and the way the data stored. It is a troublesome experiences for forensic analysts that both securing and acquiring images from an Android device which are of different versions. For example the techniques they employ to copy the images from a TMobile G1 running Android 1.5 is different from the same device running Android 1.6 or the different kernel. This is imaginable as with more number of manufacturers with over 400 Android devices, six major releases and hundreds of minor releases the forensic analysis may vary vastly. Each manufacturer may have their own specific set of drivers and softwares.

3.4.2) How Android encryption works

Disk encryption on Android is based on dm-crypt, which is a kernel feature that works at the block device layer. Therefore, it is not usable with YAFFS, which talks directly to a raw Nand flash chip, but does work with emmc and similar flash devices which present themselves to the kernel as a block device. The current preferred filesystem to use on these devices is ext4, though that is independent of whether encryption is used or not.

While the actual encryption work is a standard linux kernel feature, enabling it on an Android device proved somewhat tricky. The Android system tries to avoid incorporating GPL components, so using the cryptsetup command or libdevmapper were not available options. So making the appropriate ioctl calls into the kernel was the best choice. The Android volume daemon (vold) already did this to support moving apps to the SD card, so we chose to leverage that work for whole disk encryption. The actual encryption used for the filesystem for first release is 128 AES with CBC and ESSIV: SHA256. The master key is encrypted with 128 bit AES via calls to the openssl library.

Once it was decided to put the smarts in vold, it became obvious that invoking the encryption features would be done like invoking other vold commands, by adding a new module to vold (called cryptfs) and teaching it various commands. The commands are checkpw, restart, enablecrypto, changepw and cryptocomplete.

The other big issue was how to get the password from the user on boot. The initial plan was to implement a minimal UI that could be invoked from init in the initial ramdisk, and then init would decrypt and mount /data. However, the UI engineer said that was a lot of work, and suggested instead that init communicate upon startup to tell the framework to pop up the password entry screen, get the password, and then shutdown and have the real framework started. It was decided to go this route, and this then led to a host of other decisions described below. In particular, init set a property to tell the framework to go into the special password entry mode, and that set the stage for much communication between vold, init and the framework using properties.

Finally, there were problems around killing and restarting various services so that /data could be unmounted and remounted. Bringing up the temporary framework to get the user password requires that a tmpfs /data filesystem be mounted, otherwise the framework will not run. But to unmount the tmpfs /data filesystem so the real decrypted /data filesystem could be mounted meant that every process that had open files on the tmpfs /data filesystem had to be killed and restarted on the real /data filesystem. This magic was accomplished by requiring all services to be in 1 of 3 groups: core, main and late_start. Core services are never shut down after starting but main services are shut-down and then restarted after the disk password is entered. Late_start services are not started until after /data has been decrypted and mounted. The magic to trigger these actions is by setting the property vold.decrypt to various magic strings. Also, a new init command "class_reset" was invented to stop a service, but allow it to be restarted with a "class_start" command. If the command "class_stop" was used instead of the new command "class_reset" the flag SVC_DISABLED was added to the state of any service stopped, which means it would not be started when the command class_start was used on its class.

3.4.3) Android rooting

Android Rooting is the process of allowing users of Smartphones, tablets, and other devices running the Android mobile operating system to attain privileged control (known as "root access") within Android's subsystem.

Rooting is often performed with the goal of overcoming limitations that carriers and hardware manufacturers put on some devices, resulting in the ability to alter or replace system applications and settings, run specialized applications that require administrator-level permissions, or perform other operations that are otherwise inaccessible to a normal Android user. On Android, rooting can also facilitate the complete removal and replacement of the device's operating system, usually with a more recent release of its current operating system.

As Android derives from the Linux kernel, rooting an Android device is similar to accessing administrative permissions on Linux or any other Unix-like operating system such as FreeBSD or OS X.

Root access is sometimes compared to jailbreaking devices running the Apple iOS operating system. However, these are different concepts. In the tightly-controlled iOS world, technical restrictions prevent

- ✓ **installing or booting into a modified or entirely new operating system** (a "locked bootloader" prevents this),
- ✓ **sideloading unsigned applications onto the device, and**
- ✓ **user-installed applications from having root** privileges (or from running outside a secure sandboxed environment).

By passing all these restrictions together constitute the expansive term "jailbreaking" of Apple devices. That is, jailbreaking entails overcoming several types of iOS security features simultaneously. By contrast, only minorities of Android devices lock their bootloaders—and many vendors such as HTC, Sony, Asus and Google explicitly provide the ability to unlock devices, and even replace the operating system entirely. Similarly, the ability to sideload applications is typically permissible on Android devices without root permissions. Thus, primarily the third aspect of iOS jailbreaking, relating to superuser privileges, correlates to Android Rooting.

Description

Rooting lets all user-installed applications run privileged commands typically unavailable to the devices in the stock configuration. Rooting is required for more advanced and potentially dangerous operations including modifying or deleting system files, removing carrier- or manufacturer-installed applications, and low-level access to the hardware itself (rebooting, controlling status lights, or recalibrating touch inputs.) A typical rooting installation also installs the Superuser application, which supervises applications that are granted root or superuser rights. A secondary operation, unlocking the device's bootloader verification, is required to remove or replace the installed operating system. In contrast to iOS jailbreaking, rooting is not needed to run applications distributed outside of the Google Play Store, sometimes called *sideloading*. The Android OS supports this feature natively in two ways: through the "Unknown sources" option in the Settings menu and through the Android Debug Bridge.

However some carriers, like AT&T, prevent the installation of applications not on the Store in firmware, although several devices (including the Samsung Infuse 4G) are not subject to this rule and AT&T has since lifted the restriction on several older devices. As of 2012 the Amazon Kindle Fire defaults to the Amazon Apple store instead of Google Play, though like most other Android devices, Kindle Fire allows sideloading of applications from unknown sources and the "easy installer" application on the Amazon Apple store makes this easy. Other vendors of Android devices may look to other sources in the future. Access to alternate applications may require rooting but rooting is not always necessary. Rooting an Android phone lets the owner modify or delete the system files, which in turn lets them perform various tweaks and use apps that require root access.

Process

The process of rooting varies widely by device, but usually includes exploiting a security bug(s) in the firmware (i.e. in Android) of the device, and then copying the su binary file to a location in the current process's PATH (e.g. /system/xbin/su) and granting it executable permissions with the chmod command. A supervisor application like SuperUser or SuperSU can regulate and log elevated permission requests from other applications. Many guides, tutorials and automatic processes exist for popular Android devices facilitating a fast and easy rooting process.

For example, shortly after the HTC Dream (HTC G1) was released, it was quickly discovered that anything typed using the keyboard was being interpreted as a command in a privileged (root) shell. Although Google quickly released a patch to fix this, a signed image of the old firmware leaked, which gave users the ability to downgrade and use the original exploit to gain root access. Once an exploit is discovered, a custom recovery image that skips the digital signature check of a firmware update package can be flashed. In turn, using the custom recovery, a modified firmware update can be installed that typically includes the utilities (for example the *Superuser* application) needed to run applications as root.

The Google-branded Android phones, the Nexus One, Nexus S, Galaxy Nexus, Nexus 4 and Nexus 5 as well as their tablet counterparts, the Nexus 7 and Nexus 10, can be boot-loader unlocked by simply connecting the device to a computer while in boot-loader mode and running the Fastboot program with the command "fastboot oem unlock". After accepting a warning, the boot-loader is unlocked, so a new system image can be written directly to flash without the need for an exploit.

In 2011, Motorola, LG Electronics and HTC added security features to their devices at the hardware level in an attempt to prevent users from rooting retail Android devices. For instance, the Motorola Droid X has a security boot-loader that puts the phone in "recovery mode" if a user loads unsigned firmware onto the device, and the Samsung Galaxy S II displays a yellow triangle indicator if the device firmware has been modified.

3.4.4) Vulnerabilities

Taking specific malware out of the equation, what are some of the threats/vulnerabilities on Android devices that might be cause for concern? These certainly are not comprehensive, but do cover a significant range of the vulnerabilities and risks that may be exploited on the Android OS:

User as admin

Install apps, grant app permissions, download data, and access unprotected networks - The user can reign free over their Android domain without restriction.

The Android Market

Google's verification processes for applications entering their market have been shown to be woefully lacking over the last year or two, leading to a number of malware-infected apps and games being made legitimately available to users.

Gateway to PC

HTC devices have long been able to utilize a VPN, but increasingly other applications are becoming available for remote access – Go to Meeting, TeamViewer, Remote Rackspace. Although secured, these third party services still provide a line in to the corporate network and may be implemented fairly easily on to an endpoint.
Any Android device can be connected to a PC via a USB cable, laying out the contents of its SD card for read/write/delete. The SD card itself as removable storage can be easily accessed directly as well. Indeed these methods could be utilized themselves for bringing malware in to a corporate network, for downloading malicious content on to a PC or sucking up data as soon as it is connected.

Application permissions

In the form of a pop up, the user may see these notifications as a nuisance, a delay in accessing the newly downloaded Angry Birds levels. Or they may simply not understand the nature of the requests. Common permissions that may (read: should!) raise an eyebrow would include 'Read/Send SMS', 'Access Fine Location', 'Access IMEI, phone identity', 'Brick' (required to disable the device in trace and wipe apps), 'Access camera', and so on. Such requests may be integral to functionality, but could equally be recording calls and transmitting
sign-in credentials.

Malicious application injections

Data/process transfers between virtualized application environments are handled by a protocol of implicit and explicit intents. Transmission or interception of intent by a malicious application can result in data being compromised as the target app will respond to the string, potentially resulting in data loss.

Third party applications

One of the great things about Android is choice in terms of standard functionality, such as address books, messaging, keyboards, etc. I'm sure no one in the information security industry would need an explanation as to why it might not be a good idea to use an un-trusted third party keyboard or password manager.
Even reputable services can get mobile applications wrong, both Facebook and Twitter transmit mobile app data in the clear, i.e. without encryption, on nearly all devices. This happens despite the development of such security measures for web app versions.

Rooting

Rooting an Android device is akin to jail-breaking an iPhone, it opens out additional functionality and services to users. The process of gaining root access, requires the device to be switched from S-On to S-Off (where S =security). Additionally, root is a common exploit used by malicious applications to gain system-level access to our Android. DroidKungFu is one such threat that can root a system and install applications at that level, it escapes detection by utilizing encryption and decryption to deliver a payload.

Wi-Fi

The vulnerability of Android devices running 2.3.3 to compromise on unprotected Wi-Fi networks apparently came as a surprise to many11 – it shouldn't have, when is this practice ever safe? Beyond highlighting the need for better consumer security awareness, it leads to some other considerations around secure Wi-Fi access. Ideally sign in credentials should always be completed over a secured network, but sometimes this isn't enough. FaceNiff is an easily downloadable application that allows the user to intercept the social networking logins of any Android on their network. The only way this exploit won't work is if the user is utilizing SSL.

Remote installation

To users, the Android Web Market is a blessing. Its introduction meant a much more accessible and digestible platform for discovering the latest applications. Additionally there's no need to go back to your device when you find something you want, you can simply install it remotely from a PC. Third party Android market, AppBrain, also offers a similar service (and actually beat Google to it in the first place).

All versions of Android were at one point vulnerable to the remote writing of malicious JavaScript to the SD card through accessing an infected web page – an html download does not prompt for user confirmation on the OS, it simply happens. This is now restricted to versions 2.2 and below as the issue was addressed by Google in the Gingerbread (2.3) update. For devices still operating versions with this vulnerability, using Opera Mobile instead of the default web browser will trigger a confirmation when such download attempts begin, allowing the user to deny access.

Privacy

By default, HTC devices have geo-tag photos and Tweets. This is the primary issue with Android as a consumer device –functionality over security. Other applications claiming localized services could utilize GPS permissions for location tracking.

Manufacturer trust

Whatever their intentions, manufacturers play a significant role in user privacy. Uncovered recently is an application that sits at root level on new HTC devices (Evo, Evo 3D, Thunderbolt, Sensation) which collects and transmits a range of information on users including accounts, phone numbers, SMS, system logs, GPS locations, IP addresses, and installed apps. It is bad enough that HTC feel it is appropriate to collect and use this data without notifying the user, it is even worse that it failed to secure it! Consequently any app with the 'Access internet' permission can access this data.

Figure 37 Vulnerabilities

3.4.5) Security measures

So now we know a little more about how an Android might be compromised, we can understand about some of the controls and mitigation processes we can take to protect the device and the information it may access. Some of this advice could be implemented by best practice, policy, or a user toolkit within the enterprise, but roll out to individual user devices should also be considered.

Root and S-off

An illogical suggestion, surely S-off isn't going to further security on a device? I argue otherwise, yes rooting a device opens up the root level to access, but it also allows the user to exercise control over it. Upon completion of the root process, the flashed Clockwork Recovery Mod will install the SuperUser application. SuperUser notifies on all requests for root access, asking for authentication of the process. Immediately threats such as DroidKungFu become less intimidating, it can do very little at a root level if that root is controlled and monitored.

Permissions management

Many of the attack vectors and vulnerabilities which might be considered feasible on an Android device rely on being able to overcome the OS sandboxing. The problem is, the user has only two options when they install new applications – accept the permissions and allow the install, or reject them and don't get the application.

LBE Privacy Guard acts as somewhat of an application firewall and can solve this problem by granting the user the ability to block an application's individual permissions. I might be happy for a game to have internet access, but I can then block other permissions I feel unnecessary such as sending SMS or reading my phone number. The application will only work on rooted devices, but is one of the most effective security measures you can take on Android. You can set permissions, and will be notified of any requests that you haven't already specified a preference for. Additionally the application keeps a full security log of every permission request made by every application – a real eye-opener and the first example of SIEM on Android.

Manufacturer bootloggers

The issue with HTC devices logging and transmitting user data is a significant vulnerability and the only 'out of the box' option is to wait for a patch. This simply isn't good enough. If we are using a rooted device we have two options: 1) flash a non-stock ROM such as CyanogenMod, 2) navigate to system/app and delete the file com.htc.loggers.apk

For now this is as good as it gets for a fix. Even if we run this, it is not to say that the problem is totally solved. There are other preloaded HTC system apps which raise an eyebrow, for example androidvncserver.apk which is a remote access tool – it could easily be innocuous or tied to functionality such as trace and wipe, but it is certainly something worth being aware of.

Trace and wipe

The Android equivalent of business continuity implementation. If your Android device is lost or stolen, you can use these applications to remotely ping the device for its location and/or instruct it to delete specific content. Manufacturers such as HTC provide such measures on their devices, but third parties and AV vendors provide similar services also. The only problem with a great number of these is the ability to prevent them – if you remove the SIM, wipe the phone, or kill the data connection there is nothing you can do. Taking out some of these 'trace and wipe' apps really is as easy as uninstalling them from the Application Management menu. Any solutions on that basis should be manufacturer implemented or be at root level, e.g. TheftAware, to prevent them from being bypassed. TheftAware also does the obvious and conceals its identity with an innocuous name designated by the user; the hidden application remains visible but concealed until it is activated, at which point it disappears from the front end altogether.

Device locking

The reason for implementing some level of screen lock security on a smart phone should be pretty obvious to even the most ley user. Such a mass of personal data and material simply cannot be left on a table, available to all who are willing to swipe to unlock. Off the shelf Android addresses the issue with two propositions – a PIN or familiar pattern (a la iPhone and GrIDsure). Beyond the usual social engineering and finger smear tracing type exploits, this does a pretty competent job of preventing an unattended device relating in data loss. Other such solutions are available from the Market, with the most inventive of these being Hidden Lock. This app works by returning your locked device to the previous screen but with all the functions blocked – the device can only then be unlocked by correctly selecting the position of an invisible padlock.

Securing data

Firstly all the usual security rules need to be exercised, e.g. only connecting to trusted, secure networks. Beyond this though, the user needs to consider what the appropriate security measure is for whatever they're doing – encryption for data transmission, VPN for secure connection.

WhisperCore provides full desk encryption and also retains an encrypted backup of all data on the device. The system is designed to be unobtrusive, keeping impact on users to a minimum. It encrypts at 256 bit AES. This is a great way to protect against data loss in the event of a stolen or missing handset, if trace and wipe fails, you can rest safe in the knowledge that any sensitive information or business assets are locked down. All we want from users is to keep their encryption keys safe.

With encryption implemented, mobile data is secure at rest. In order to keep it secure in transfer, then a VPN solution should be considered. This may be manufacturer specific (HTC), from a reputable security vendor (Astaro), or a specialist start up (TunnelDroid). Encryption can be enabled as part of the VPN client built in to the OS, although a more controllable solution provides more powerful protection.

With products like Zenprise on the horizon, it won't be long before we see proper DLP on an Android – the company claim that users will be able to view but not send out protected content, and any

DLP-protected documents will have a built-in expiry. It has also been announced in the last few days that Symantec will soon be throwing its proverbial hat in to the ring with a DLP solution for mobile devices.

Installing trusted packages

Given the ability to install non-Market applications on to a Google device off the bat (unlike iPhone where jailbreaking is required), and the recent spate of malicious apps on the Android Market, it might be time to consider verifying contents of APK files. For the uninitiated, APK files are the standard Android install file format and are a variant of JAR. A program called APK Inspector has recently been released that will scan the assets, resources, and certificates contained within the APK to ensure it is secure. This is available via http://code.google.com/p/apkinspector and could be utilized as part of the application install process to ensure all packages are verified before being flashed to the phone.

Anti-virus

Shutting the door after the proverbial horse has vaulted, anti-virus by itself is a great way to do security wrong. It is however a useful failsafe and a must as part of a comprehensive security solution.
The lack of malware signatures, hardware limitations, and an uncertain market potential in the early days of Android meant that the big players were initially reluctant to enter in to the space with bespoke solutions. This left a gap for smaller niche players to gain some degree of market share. Indeed even now mobile specific AV is available from some of the larger vendors only as part of a wider 'all device' package. Two of the most reputable AV packages on Android to date have been from start ups, and in the form of free download. The free price point helped DroidSecurity grow its install base to 4.5m users rapidly and led to its subsequent acquisition for $4.1m by AVG. Another popular

Android security vendor is Lookout which has recently raised $40m in its last round of VC funding. In the meanwhile all the big security players have rushed to get their solutions to market, and so recently we have seen the likes of Norton, Kaspersky, and Trend Micro enter the space with offerings. Many of these applications are fairly comparable in terms of performance and detection, although there are several issues with the concept itself.

Firstly is the problem of where the AV scans, it covers the apps folders, SD card, SMS, and contacts typically. None of these apps are asking for root access, and therefore they are failing to search for infections on the area of the device that is most targeted and vulnerable. With no root integration how useful can mobile antivirus be? It is comparable to only scanning Documents & Settings on a PC, ignoring altogether Program Files and System folders. This coupled with the fact that many of the household names' Android AV packages are readily available in cracked versions on warez sites, is a significant cause for concern. If a security application cannot maintain its own integrity, how effective can it be? Indeed there is even talk of an exploit on the OS that allows AV to be bypassed and deactivated altogether. We come back to where we started then AV on Android must be part of a wider security solution.

Authentication

Much of the mention of authentication around Android comes in the form of using the devices as a means of authentication, like an OTP or token. What is often overlooked is authentication on the device itself. Google has recently taken to factoring this in to the latest update to the Market (for those fortunate enough to get a push update or who have managed to do this manually), the user can now set a password which must be entered before any purchases are possible. Given that this is now essentially baked in to a system app, how feasible would it be for Google to implement this in to the Android framework. That way password protection can be integrated in to other processes such as updating corporate email, posting to company Twitter accounts, and so on. This is assuming of course that going forward, Chocolate Mountain is able to prevent access to its authentication tokens – something it has failed on already this year.

Some manufacturers are now starting to implement biometric authentication in to devices such as the Motorola Atrix, and a number of iris and fingerprint recognition applications are available on the Market. As with all personal data and these physical traits are still information, vendor trust is imperative and the same questions that all security suppliers should expect are applicable: Who holds the data? How is it stored? Who has access to it, is it shared with 3rd parties? Is it destroyed if you opt out of the service? As secure a measure as biometric authentication can be, smaller suppliers must be scrutinized, if the technology becomes more widely used in the future then you will never be able to change your 'password' if it is compromised.

Link security

Users love to click links. Whether it be in a phishing mail/pop-up or on a friend's Facebook wall, malicious links are always loitering in the background waiting to seduce and ensnare hapless users. Awareness should cure this, but it's one of those on-going struggles that I'm sure will be familiar to many internal information security departments. There are a number of vendors that have created link security applications for PC, and there are options available on Android too, albeit in fewer numbers. AVG with its Mobilation app and Lookout Mobile
Security seems to be leading the way here, providing a solution for secure browsing. With AV limited on the Android platform, this holistic approach to security is certainly reassuring, and more vendors should be looking to integrate as much functionality in to their Android suites as possible.

Email security and mobile device management

How do you let mobile users access their corporate email on an Android device? Is it better practice to set up and secure an Active Exchange sync on the default mobile app, or encourage the use of a web service such as Mimecast or OWA? To-date, Exchange is really only doable as an option on Android out-of-the-box through an HTC device, or use of a Sense-based ROM (the latter requiring root). The HTC Mail app can be set to automatically use an encrypted SSL connection for any transfer of email data. This is the only way to comply with the security protocols on Exchange, without sending encrypted messages through a VPN using another application.

Mobile Device Management (MDM) and Mobile Email Gateway (MEG) products from the likes of Good Technology, AirWatch, and Echoworx allow enterprises to roll out email security solutions that are compliant to industry standards such as SOX and HIPAA. Such packages monitor, securely configure, and enforce policy over the air. Tools within MDMs such as application management and blacklisting introduce effective controls, and key system data can be accessed. One feature of such products that does leave some area for questioning is the decision to blacklist and disconnect a rooted device. This decision to exclude the root level from considerations is reliant upon the on-going assumption that a rooting process must occur for a root exploit to run.

There is significant value in utilizing an MDM or MEG as an additional layer of protection between devices and servers. This technology is still somewhat in its infancy and so there are limitations. One such shortcoming is around the range of supported devices, this is fine if it is a business phone, a compatible device can be selected and rolled out by the organization, however personal devices may cause issues here. Furthermore the question of ownership creates another barrier to implementation of these products – will users accept this on a personal device, especially considering some of the measures include blocking applications, tracking GPS location, and remotely locking and wiping the handset.

Of course there is always the problem of corporate email accounts sitting open on mobile devices, but hopefully users will implement other measures that mean this isn't possible. Certainly one such step that should be taken is to direct the attachment cache to store to the device's memory as opposed to the SD card, ensuring that data loss isn't quite as easy as plugging and playing a micro SD. Additionally there are other controls which could be considered - notifications can be disabled and the icon and name of the app on the menu could be edited to hide the asset; it is possible to limit the number of days displayed in the device inbox, here a trade off must be made between enabling mobile working and implementing effective security in the event of a misplaced or stolen smart phone.

Firewalls

Really LBE Privacy Guard should be helping in scratching that firewall itch on Android, but the data that is transmitted from your smart phone should be monitored as well. This may seem superfluous but given that iPhone users' location data is tracked and sent unencrypted via iTunes, this might not seem like such an unnecessary measure. Whisper Core Systems, which brought FDE to Android, also offers Whisper Monitor. All outbound data is monitored and the user is notified of any anomalies depending on the rule sets and exceptions that are set up. Furthermore, if there is any traffic causing concern, URLs can be blocked and ports closed via the app to prevent the process from continuing to run its course.

Secure storage and NFC payments

There are numerous applications available via the Market offering secure wallets/folders and password management. This concept offers a theoretically secure and password-protected environment for sensitive assets. However there is always the issue of vendor trust when it comes to utilizing such applications to protect information. Of course when Google Wallet rolls out properly to more than one Android device (i.e. Nexus S) it should theoretically prove a logical solution to secure storage, however it will also open up the potential implications of a security breach or lost device significantly. Losing a Smartphone is already caused enough for concern, but going forwards there will be more significant financial implications, as the emergence of NFC-enabled devices will mean that this is now akin to losing a debit card or wallet. Inevitably the security controls Google implements around this transaction process and its Wallet application will be carefully scrutinized, not just from an assurance stand point, but also in their impact on user experience and speed of transaction. With the roll out at such early stages it is difficult to suggest how to protect these specific assets, however with all things security, best practice will only ever serve you in good stead.

Figure 38 Security measures

3.4.5.1) User security features

File system Encryption

Android 3.0 and later provides full file system encryption, so all user data can be encrypted in the kernel using the decrypt implementation of AES128 with CBC and ESSIV: SHA256. The encryption key is protected by AES128 using a key derived from the user password, preventing unauthorized access to stored data without the user device password. To provide resistance against systematic password guessing attacks (e.g. “rainbow tables” or brute force), the password is combined with a random salt and hashed repeatedly with SHA1 using the standard PBKDF2 algorithm prior to being used to decrypt the filesystem key. To provide resistance against dictionary password guessing attacks, Android provides password complexity rules that can be set by the device administrator and enforced by the operating system. Filesystem encryption requires the use of a user password, pattern-based screen lock is not supported.

Password Protection

Android can be configured to verify a user-supplied password prior to providing access to a device. In addition to preventing unauthorized use of the device, this password protects the cryptographic key for full file system encryption. Use of a password and/or password complexity rules can be required by a device administrator.

Device Administration

Android 2.2 introduces support for enterprise applications by offering the Android Device Administration API. The Device Administration API provides device administration features at the system level. These APIs allow you to create security-aware applications that are useful in enterprise settings, in which IT professionals require rich control over employee devices. For example, the built-in Android Email application has leveraged the new APIs to improve Exchange support. Through the Email application, Exchange administrators can enforce password policies — including alphanumeric passwords or numeric PINs — across devices. Administrators can also remotely wipe (that is, restore factory defaults on) lost or stolen handsets. Exchange users can sync their email and calendar data.

Here are examples of the types of applications that might use the Device Administration API:

- Email clients.
- Security applications that do remote wipe.
- Device management services and applications.

3.4.5.2) Android application security

Elements of Applications

Android provides an open source platform and application environment for mobile devices. The core operating system is based on the Linux kernel. Android applications are most often written in the Java programming language and run in the Dalvik virtual machine. However, applications can also be written in native code. Applications are installed from a single file with the .apk file extension.

The main Android application building blocks are:

- **AndroidManifest.xml**: The AndroidManifest.xml file is the control file that tells the system what to do with all the top-level components (specifically activities, services, broadcast receivers, and content providers described below) in an application. This also specifies, which permissions are required.

- **Activities**: An Activity is, generally, the code for a single, user-focused task. It usually includes displaying a UI to the user, but it does not have to -- some Activities never display UIs. Typically, one of the application's Activities is the entry point to an application.

- **Services**: A Service is a body of code that runs in the background. It can run in its own process, or in the context of another application's process. Other components "bind" to a Service and invoke methods on it via remote procedure calls. An example of a Service is a media player: even when the user quits the media-selection UI, the user probably still intends for music to keep playing. A Service keeps the music going even when the UI has completed.

- **Broadcast Receiver**: A Broadcast Receiver is an object that is instantiated when an IPC mechanism known as an intent is issued by the operating system or another application. An application may register a receiver for the low battery message, for example, and change its behavior based on that information.

The Android Permission Model: Accessing Protected APIs

All applications on Android run in an Application Sandbox. By default, an Android application can only access a limited range of system resources. The system manages Android application access to resources that, if used incorrectly or maliciously, could adversely impact the user experience, the network, or data on the device.

These restrictions are implemented in a variety of different forms. Some capabilities are restricted by an intentional lack of APIs to the sensitive functionality (e.g. there is no Android API for directly manipulating the SIM card). In some instances, separation of roles provides a security measure, as with the per-application isolation of storage. In other instances, the sensitive APIs are intended for use by trusted applications and protected through a security mechanism known as Permissions.

These protected APIs include:

- ✓ Camera functions
- ✓ Location data (GPS)
- ✓ Bluetooth functions
- ✓ Telephony functions
- ✓ SMS/MMS functions
- ✓ Network/data connections

These resources are only accessible through the operating system. To make use of the protected APIs on the device, an application must define the capabilities it needs in its manifest. When preparing to install an application, the system displays a dialog to the user that indicates the permissions requested and asks whether to continue the installation. If the user continues with the installation, the system accepts that the user has granted all of the requested permissions. The user can not grant or deny individual permissions -- the user must grant or deny all of the requested permissions as a block.

Once granted, the permissions are applied to the application as long as it is installed. To avoid user confusion, the system does not notify the user again of the permissions granted to the application, and applications that are included in the core operating system or bundled by an OEM do not request permissions from the user. Permissions are removed if an application is uninstalled, so a subsequent re-installation will again result in display of permissions.

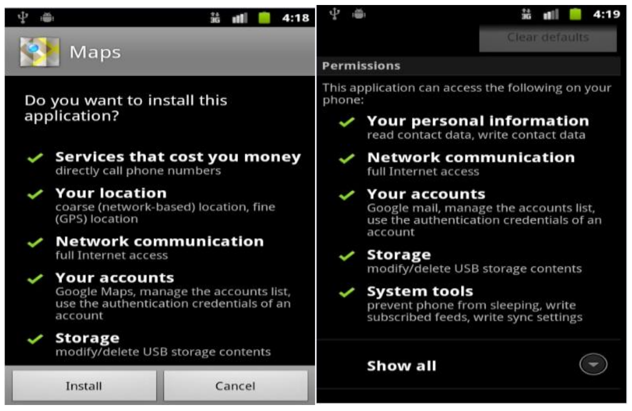
Within the device settings, users are able to view permissions for applications they have previously installed. Users can also turn off some functionality globally when they choose, such as disabling GPS, radio, or Wi-Fi.

In the event that an application attempts to use a protected feature which has not been declared in the application's manifest, the permission failure will typically result in a security exception being thrown back to the application. Protected API permission checks are enforced at the lowest possible level to prevent circumvention. An example of the user messaging when an application is installed while requesting access to protected APIs is shown in next Figure**.**

The system default permissions are described at: https://developer.android.com/reference/android/Manifest.permission.html. Applications may declare their own permissions for other applications to use. Such permissions are not listed in the above location.

When defining a permission a protection Level attribute tells the system how the user is to be informed of applications requiring the permission, or who is allowed to hold a permission. Details on creating and using application specific permissions are described at: https://developer.android.com/training/articles/security-tips.html.

There are some device capabilities, such as the ability to send SMS broadcast intents, that are not available to third-party applications, but that may be used by applications pre-installed by the OEM. These permissions use the signature or System permission.


4:18

Maps

Do you want to install this application?

✓ **Services that cost you money**
directly call phone numbers

✓ **Your location**
coarse (network-based) location, fine (GPS) location

✓ **Network communication**
full Internet access

✓ **Your accounts**
Google Maps, manage the accounts list, use the authentication credentials of an account

✓ **Storage**
modify/delete USB storage contents

Install | Cancel

4:19

Clear defaults

Permissions

This application can access the following on your phone:

✓ **Your personal information**
read contact data, write contact data

✓ **Network communication**
full Internet access

✓ **Your accounts**
Google mail, manage the accounts list, use the authentication credentials of an account

✓ **Storage**
modify/delete USB storage contents

✓ **System tools**
prevent phone from sleeping, write subscribed feeds, write sync settings

Show all

Figure 39 Display of permissions for applications

How Users understand Third-Party Applications

Android strives to make it clear to users when they are interacting with third-party applications and inform the user of the capabilities those applications have. Prior to installation of any application, the user is shown a clear message about the different permissions the application is requesting. After install, the user is not prompted again to confirm any permission.

There are many reasons to show permissions immediately prior to installation time. This is when user is actively reviewing information about the application, developer, and functionality to determine whether it matches their needs and expectations. It is also important that they have not yet established a mental or financial commitment to the application, and can easily compare the application to other alternative apps.

Some other platforms use a different approach to user notification, requesting permission at the start of each session or while applications are in use. The vision of Android is to have users switching seamlessly between applications at will. Providing confirmations each time would slow down the user and prevent Android from delivering a great user experience. Having the user review permissions at install time gives the user the option to not install the application if they feel uncomfortable.

Also, many user interface studies have shown that over-prompting the user causes the user to start saying "OK" to any dialog that is shown. One of Android's security goals is to effectively convey important security information to the user, which cannot be done using dialogs that the user will be trained to ignore. By presenting the important information once, and only when it is important, the user is more likely to think about what they are agreeing to.

Some platforms choose not to show any information at all about application functionality. That approach prevents users from easily understanding and discussing application capabilities. While it is not possible for all users to always make fully informed decisions, the Android permissions model makes information about applications easily accessible to a wide range of users. For example, unexpected permissions requests can prompt more sophisticated users to ask critical questions about application functionality and share their concerns in places such as Google Play where they are visible to all users.

4) Data acquisition

4.1) The phenomenon of data acquisition.

Data acquisition is the process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer. Data acquisition systems (abbreviated with the acronym **DAS** or **DAQ**) typically convert analog waveforms into digital values for processing. The components of data acquisition systems include:

- Sensors that convert physical parameters to electrical signals.
- Signal conditioning circuitry to convert sensor signals into a form that can be converted to digital values.
- Analog-to-digital converters, which convert conditioned sensor signals to digital values.

Data acquisition applications are controlled by software programs developed using various general purpose programming languages such as BASIC, C, Fortran, Java, Lisp, Pascal.

History

In 1963, IBM produced computers which specialized in data acquisition. These include the IBM 7700 Data Acquisition System and its successor, the IBM 1800 Data Acquisition and Control System. These expensive specialized systems were surpassed in 1974 by general purpose S-100 computers and data acquisitions cards produced by Tecmar/Scientific Solutions Inc. In 1981 IBM introduced the IBM Personal Computer and Scientific Solutions introduced the first PC data acquisition products.

Source

Data acquisition begins with the physical phenomenon or physical property to be measured. Examples of this include temperature, light intensity, gas pressure, fluid flow, and force. Regardless of the type of physical property to be measured, the physical state that is to be measured must first be transformed into a unified form that can be sampled by a data acquisition system. The task of performing such transformations falls on devices called *sensors*.

A sensor, which is a type of *transducer*, is a device that converts a physical property into a corresponding electrical signal (e.g., Strain gauge, thermistor). An acquisition system to measure different properties depends on the sensors that are suited to detect the those properties. Signal conditioning may be necessary if the signal from the transducer is not suitable for the DAQ hardware being used. The signal may need to be filtered or amplified in most cases. Various other examples of signal conditioning might be bridge completion, providing current or voltage excitation to the sensor, isolation, linearization. For transmission purposes, single ended analog signals, which are more susceptible to noise can be converted to differential signals. Once digitized, the signal can be encoded to reduce and correct the transmission errors.

DAQ Hardware

DAQ hardware is what usually interfaces between the signal and a PC. It could be in the form of modules that can be connected to the computer's ports (parallel, serial, USB, etc.) or cards connected to slots (S-100 bus, AppleBus, ISA, MCA, PCI, PCI-E, etc.) in the motherboard. Usually the space on the back of a PCI card is too small for all the connections needed, so an external breakout box is required. The cable between this box and the PC can be expensive due to the many wires, and the required shielding.

DAQ cards often contain multiple components (multiplexer, ADC, DAC, TTL-IO, high speed timers, RAM). These are accessible via a bus by a microcontroller, which can run small programs. A controller is more flexible than a hard wired logic, yet cheaper than a CPU so that it is permissible to block it with simple polling loops. For example: Waiting for a trigger, starting the ADC, looking up the time, waiting for the ADC to finish, move value to RAM, switch multiplexer, get TTL input, let DAC proceed with voltage ramp.

DAQ Software

DAQ software is needed in order for the DAQ hardware to work with a PC. The device driver performs low-level register writes and reads on the hardware, while exposing a standard API for developing user applications. A standard API such as COMEDI allows the same user applications to run on different operating systems, e.g. a user application that runs on Windows will also run on Linux.

4.1.1) The significance of the "Data Acquisition" in Smartphones.

Mobile phones are digital media. In principle, this means that mobile phones have the same evidentiary possibilities as other digital media, such as hard drives. For example it is possible to extract deleted information from a mobile phone, in the same way it is possible on a hard drive. However, mobile phones also suffer from the same evidentiary problems as other digital media. As with a computer, the content of a mobile phone is fragile and can easily be deleted and overwritten. Mobile phones should therefore be handled with great care and insight, just as any other digital media.
A long range of data (evidence items) can be found in modern mobile phones. In addition to the evidence related to the phone system itself, modern phones have other evidence items that should be mentioned:

- ✓ Images
- ✓ Sounds

- ✓ Multimedia messages
- ✓ WAP/web browser history
- ✓ Email
- ✓ Calendar items
- ✓ Contacts

Last but not least, the importance of SMS text messages should not be underestimated. The Short Message Service is very widely used, and messages are stored on the sending and receiving mobile phone. The ability to recover deleted text messages would have great value in many investigations. The investigation of possibilities to recover deleted text messages has been the main motivation for this research.

4.1.2) The significance of the "Data Acquisition" in Android OS.

Considering the unique characteristics of the Android platform and different scenarios which a forensic analyst may come across, a data acquisition method is proposed and its workflow is shown in Figure 40. In the figure, different scenarios and their respective procedures to be adopted by the analyst are presented. By using the proposed method, a forensic analyst may retrieve maximum information from the mobile device, so that the evidence may be documented, preserved and processed in the safest and least intrusive manner as possible.

Initial procedures for preservation of the data in a Smartphone

Upon receiving a Smartphone, the forensic analyst must follow the procedures in order to preserve the stored data on the seized equipment. So, he should check if the phone is turned on or not. If the phone is powered off, one should evaluate the possibility of extracting data from its memory card. It should be pointed that some Android phones have an internal memory card, so it is not possible to remove it in order to copy its data through the use of a standard USB card reader. On the other hand, if it is feasible to remove the memory card, it should be removed and duplicated to an analyst memory card to ensure its preservation. To copy data from the memory card, one may use the same approach used with pen drives. The forensic expert could use forensic tools to copy the data or even run a disk dump and then generate the hash of the duplicate data. At the end of the process, the analyst' memory card with the copy should be returned to the device.

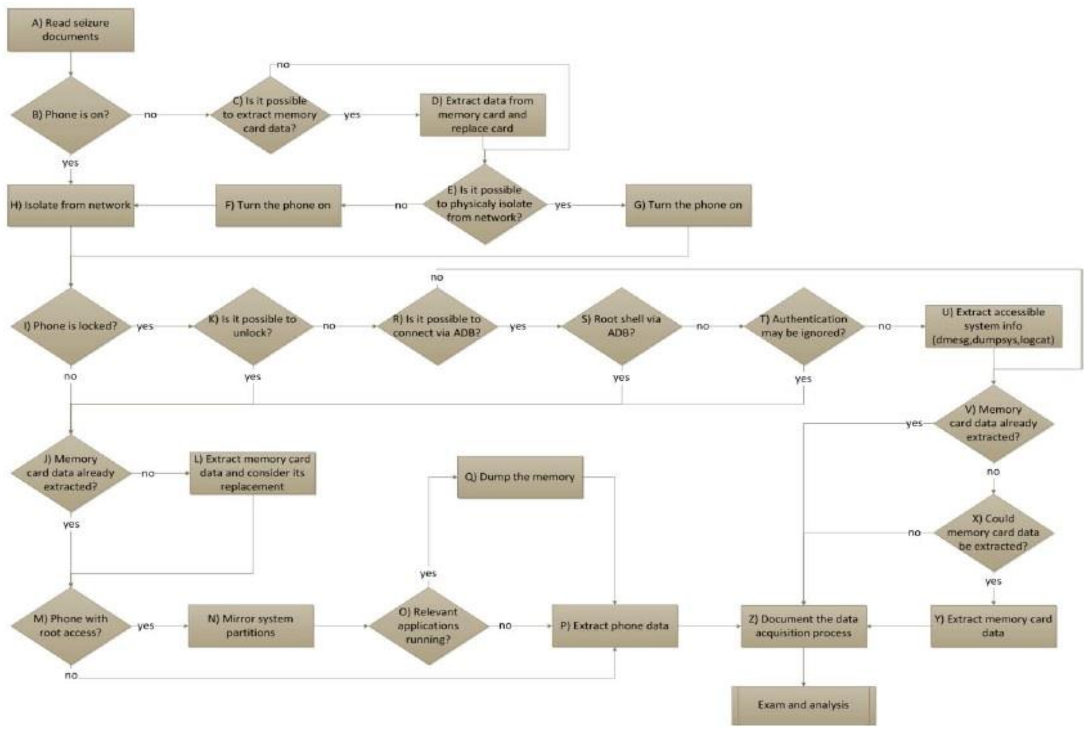

A) Read seizure documents
B) Phone is on?
no
C) Is it possible to extract memory card data?
yes
D) Extract data from memory card and replace card
no
yes
E) Is it possible to physicaly isolate from network?
yes
G) Turn the phone on
no
F) Turn the phone on
H) Isolate from network
I) Phone is locked?
yes
K) Is it possible to unlock?
no
R) Is it possible to connect via ADB?
yes
S) Root shell via ADB?
no
T) Authentication may be ignored?
no
U) Extract accessible system info (dmesg,dumpsys,logcat)
no
no
yes
yes
yes
V) Memory card data already extracted?
yes
no
J) Memory card data already extracted?
no
L) Extract memory card data and consider its replacement
yes
Q) Dump the memory
X) Could memory card data be extracted?
no
yes
yes
M) Phone with root access?
yes
N) Mirror system partitions
O) Relevant applications running?
no
P) Extract phone data
Z) Document the data acquisition process
Y) Extract memory card data
no
Exam and analysis

Figure 40 Workflow with the process of acquiring data from a Smartphone with the Android operating system.

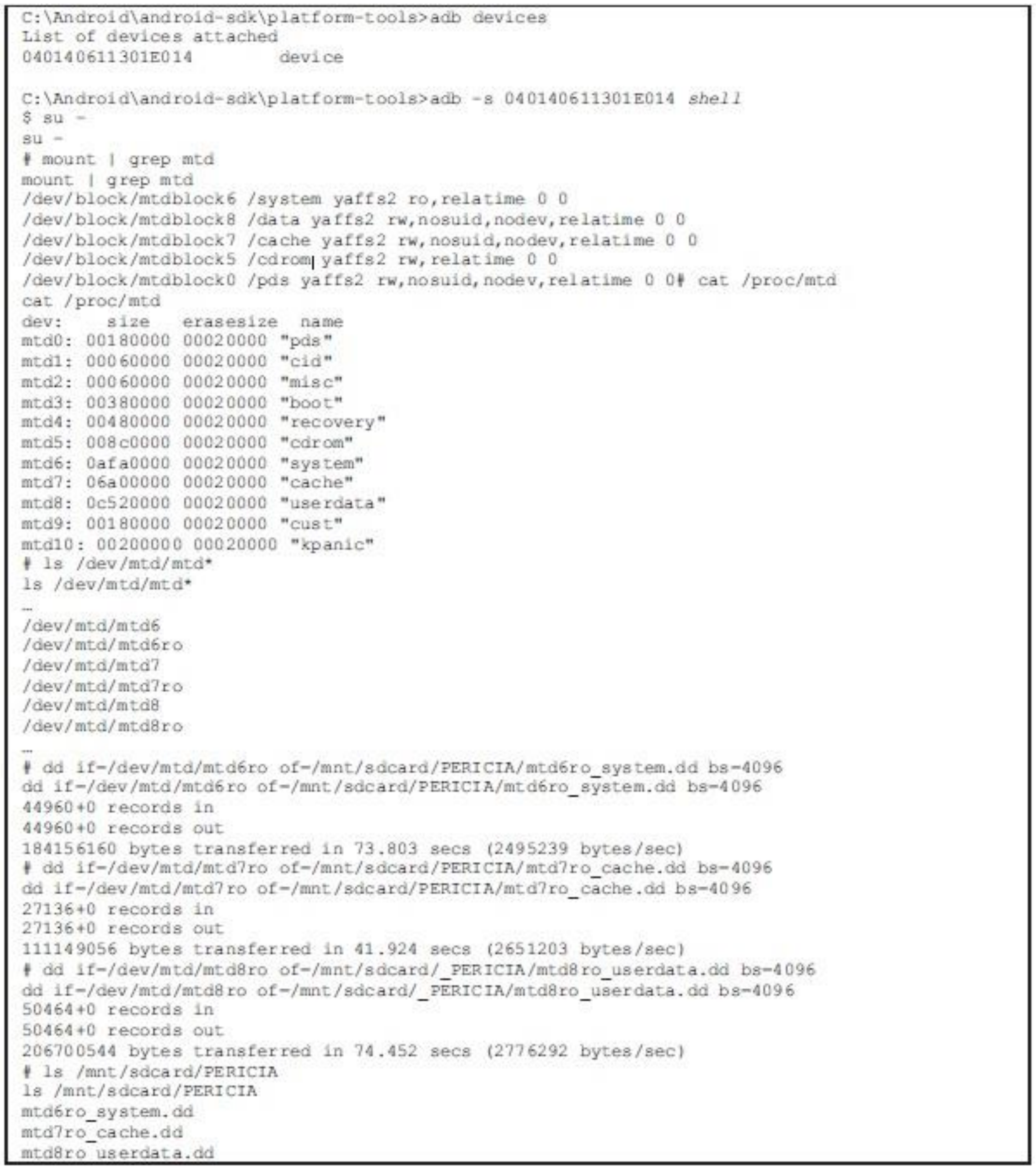
The next step is to isolate the telephone from the network. The ideal situation is to use a room with physical insulation from electromagnetic signals. However, when one does not have such an infrastructure, he should set the Smartphone to flight or offline mode. From the moment the power is on, he must immediately configure it to such connectionless mode, thus avoiding data transmission, receiving calls or SMS (Short Message Service) after the equipment seizure time. If by any chance, before it is isolated from the network, the phone receives an incoming call, message, email or other information, the analyst should document and describe it in its final report, which will be written after the data extraction, examination and analysis processes.

With the Smartphone isolated from telecommunication networks, the forensic analyst should check if the Android has been configured to provide an authentication mechanism, such as a password or tactile pattern. Afterwards, he should carry out the procedures described in the following sections, which depend on the access control mechanism which is configured on the device.

Smartphone without access control

The least complex situation that an examiner may encounter is one which the mobile is not locked and is readily able to have its data extracted. In this situation, one must first extract data from memory cards, if they have not been copied, and in case of removable memory cards, reinstall into the device the cards that have received the copies, preserving the original ones.

With the data from memory cards extracted and properly preserved, the examiner should check if the Android has super user privileges enabled. The application called "Superuser" can be installed to provide access to such privileges. From the moment the analyst is faced with an Android phone with super user privileges, he can gain access to all stored data in the device without any restriction. Using the USB debugging tool ADB (Android Debug Bridge), present in the Android SDK, one can connect to the device, access a command shell with super user privileges and make a copy of the system partitions stored in its internal memory, as illustrated in Figure xx.

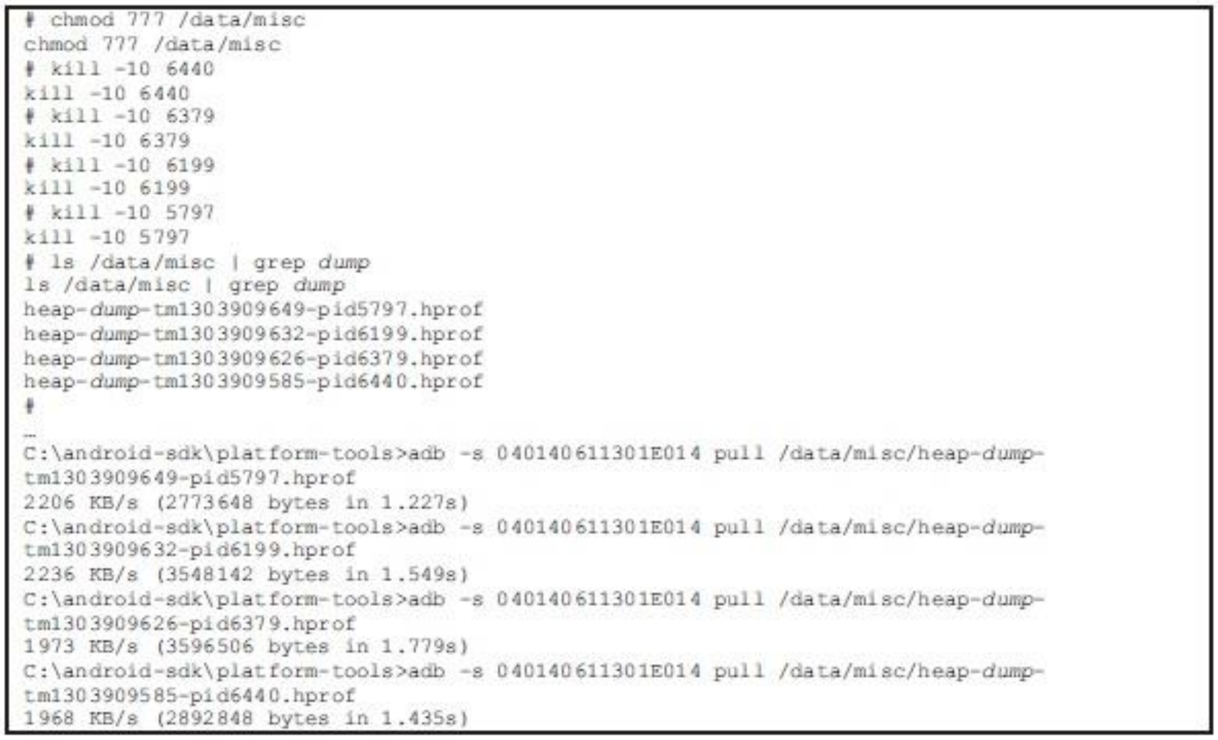


```
C:\Android\android-sdk\platform-tools>adb devices
List of devices attached
040140611301E014        device

C:\Android\android-sdk\platform-tools>adb -s 040140611301E014 shell
$ su -
su -
# mount | grep mtd
mount | grep mtd
/dev/block/mtdblock6 /system yaffs2 ro,relatime 0 0
/dev/block/mtdblock8 /data yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock7 /cache yaffs2 rw,nosuid,nodev,relatime 0 0
/dev/block/mtdblock5 /cdrom yaffs2 rw,relatime 0 0
/dev/block/mtdblock0 /pds yaffs2 rw,nosuid,nodev,relatime 0 0# cat /proc/mtd
cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00180000 00020000 "pds"
mtd1: 00060000 00020000 "cid"
mtd2: 00060000 00020000 "misc"
mtd3: 00380000 00020000 "boot"
mtd4: 00480000 00020000 "recovery"
mtd5: 008c0000 00020000 "cdrom"
mtd6: 0afa0000 00020000 "system"
mtd7: 06a00000 00020000 "cache"
mtd8: 0c520000 00020000 "userdata"
mtd9: 00180000 00020000 "cust"
mtd10: 00200000 00020000 "kpanic"
# ls /dev/mtd/mtd*
ls /dev/mtd/mtd*
…
/dev/mtd/mtd6
/dev/mtd/mtd6ro
/dev/mtd/mtd7
/dev/mtd/mtd7ro
/dev/mtd/mtd8
/dev/mtd/mtd8ro
…
# dd if-/dev/mtd/mtd6ro of-/mnt/sdcard/PERICIA/mtd6ro_system.dd bs-4096
dd if-/dev/mtd/mtd6ro of-/mnt/sdcard/PERICIA/mtd6ro_system.dd bs-4096
44960+0 records in
44960+0 records out
184156160 bytes transferred in 73.803 secs (2495239 bytes/sec)
# dd if-/dev/mtd/mtd7ro of-/mnt/sdcard/PERICIA/mtd7ro_cache.dd bs-4096
dd if-/dev/mtd/mtd7ro of-/mnt/sdcard/PERICIA/mtd7ro_cache.dd bs-4096
27136+0 records in
27136+0 records out
111149056 bytes transferred in 41.924 secs (2651203 bytes/sec)
# dd if-/dev/mtd/mtd8ro of-/mnt/sdcard/_PERICIA/mtd8ro_userdata.dd bs-4096
dd if-/dev/mtd/mtd8ro of-/mnt/sdcard/_PERICIA/mtd8ro_userdata.dd bs-4096
50464+0 records in
50464+0 records out
206700544 bytes transferred in 74.452 secs (2776292 bytes/sec)
# ls /mnt/sdcard/PERICIA
ls /mnt/sdcard/PERICIA
mtd6ro_system.dd
mtd7ro_cache.dd
mtd8ro_userdata.dd
```

Figure 41 Commands to list connected devices, display partition information, and generate the partitions dump.

It should be pointed that, by carrying out procedure described in Figure 2, the mirrored partition images will be written to the memory card which is installed in the device. In some situations, it may not be possible to replace the original memory card by an analyst one. Nevertheless, regardless of its replacement, removable media's data must have been mirrored prior to the system mirroring and copying. By doing that, data stored in the original memory card, seized with the Smartphone, are preserved and the forensic expert should point that in his report that will be produced by the end of data analysis.

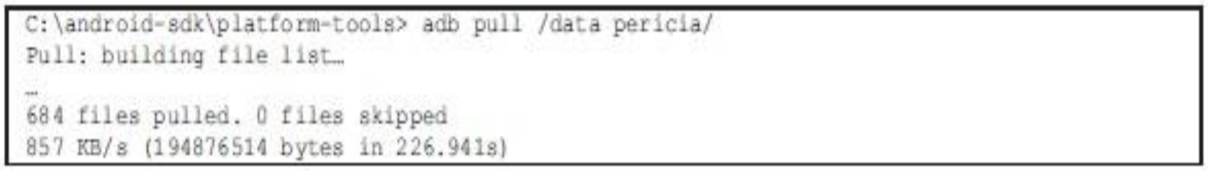
After mirroring the partitions, one should observe the running processes and assess the need to get run-time information, which is loaded in the device's memory. Hence, it is possible to extract memory data used by running applications to access sensitive information, such as passwords and cryptographic keys. By using a command shell with super user credentials, the “/data/misc” directory's permissions must be changed. Afterwards, one must kill the target running process, so that a memory dump file for the killed process is created (Cannon, T., 2010). Data extraction of a telephone with available "super user" credentials may be finished in this moment. Figure xxx displays the technique described by Thomas Cannon (Cannon, T., 2010).



```
# chmod 777 /data/misc
chmod 777 /data/misc
# kill -10 6440
kill -10 6440
# kill -10 6379
kill -10 6379
# kill -10 6199
kill -10 6199
# kill -10 5797
kill -10 5797
# ls /data/misc | grep dump
ls /data/misc | grep dump
heap-dump-tm1303909649-pid5797.hprof
heap-dump-tm1303909632-pid6199.hprof
heap-dump-tm1303909626-pid6379.hprof
heap-dump-tm1303909585-pid6440.hprof
#
…
C:\android-sdk\platform-tools>adb -s 040140611301E014 pull /data/misc/heap-dump-
tm1303909649-pid5797.hprof
2206 KB/s (2773648 bytes in 1.227s)
C:\android-sdk\platform-tools>adb -s 040140611301E014 pull /data/misc/heap-dump-
tm1303909632-pid6199.hprof
2236 KB/s (3548142 bytes in 1.549s)
C:\android-sdk\platform-tools>adb -s 040140611301E014 pull /data/misc/heap-dump-
tm1303909626-pid6379.hprof
1973 KB/s (3596506 bytes in 1.779s)
C:\android-sdk\platform-tools>adb -s 040140611301E014 pull /data/misc/heap-dump-
tm1303909585-pid6440.hprof
1968 KB/s (2892848 bytes in 1.435s)
```

Figure 42 Presentantion of commands for altering a directory's permissions, killing processes to create processes' memory.

It is noteworthy that, in order to inspect the acquired data, the analyst should have an examination environment with tools that are capable of mounting images having the device's file system, which is in most cases the YAFFS2. The technique described by Andrew Hoog may be used to examine that file system (Hoog, A., 2011). Nevertheless, it is recommended that a logical copy of system files is made directly to the analyst's workstation, as shown in Figure xxx.



```
C:\android-sdk\platform-tools> adb pull /data pericia/
Pull: building file list…
…
684 files pulled. 0 files skipped
857 KB/s (194876514 bytes in 226.941s)
```

Figure 43 Copy of logical files stored in the device's "/data" directory to the "pericia" directory in the analyst's

The stored data in the "/data" directory, for instance, contain information regarding the installed applications and system configuration. Logical copy of files will create redundancy that may be useful during the examination phase, especially in situations when it is not necessary to delve into system partitions. In addition, some applications may be active in the system, in such a way that a simple visual inspection may provide information which would be difficult to access by means of analyzing the created image. Moreover, forensic extraction tools may be used to interpret stored data.

In situations when the Smartphone "super user" privileges are not available, data extraction from its internal memory should be carried out by visually inspecting and navigating through the device graphic user interface. Alternatively, forensic tools and applications may be used to assist the analyst in extracting device's data. Nevertheless, it is important to check the information gathered by such tools, because the Android OS has different versions, as well as manufacturer and telephone carrier customizations, which may interfere in the automated tools' proper functioning. That is numerous applications that may store meaningful information for an investigation, whose data extraction is not supported by forensic tools. It is clear that the forensic analyst needs to have proper knowledge regarding the Android platform and its applications, since relevant information extraction should be conducted in the most complete way.

Some Android Smartphones allow that their internal memories must be copied using boot loader or recovery partitions vulnerabilities, without having "super user" credentials. It is up to the analyst to evaluate if it is possible and viable to apply such techniques for that kind of device. It is suggested that the investigation team should discuss the need of using such techniques and consider the risks and impacts to the exam results.

Regarding existing forensic tools, the viaForensics company developed a free tool to law enforcement agencies called “Android Forensic Logical Application” (AFLogical) (ViaForensics, 2011), whose goal is to extract information from Android Smartphones. In addition, recently the commercial tool viaExtract was released and, according to viaForensics, it has more consistent and relevant features, such as generating reports. Another very useful tool is the "Cellebrite UFED", whose version 1.1.7.5, released in July of 2011, carry out physical extraction from a few models without the need of "super user" privileges. The same tool has a plugin to view Android's SQLite databases and other application files, such as Gmail, SMS, MMS and contacts.

Smartphone with access control

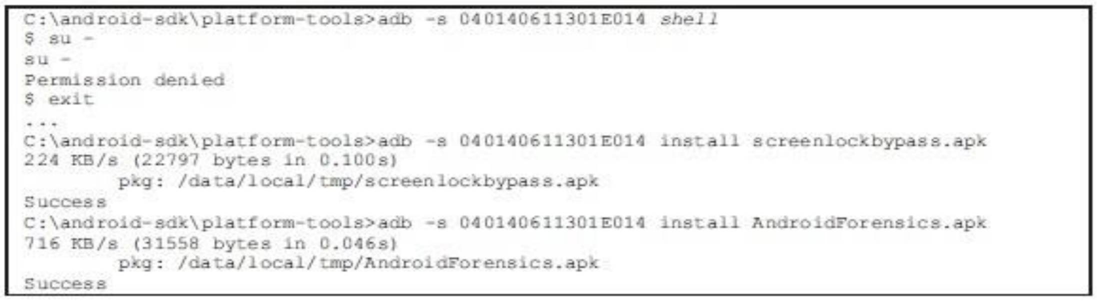
In the likely event the Android Smartphone has access control, such as a password or tactile pattern, there are still techniques to be used to access the device.

According to NIST (Jansen, W., Ayers, R., 2007), there are three ways of gaining access to locked devices. The first one is the investigative method, whereby the researcher seeks possible passwords in the place where the Smartphone was seized or even interview the its alleged owner so that he cooperates voluntarily by providing his password. Another way is to gain access via hardware, when the analyst performs a research on that specific given model to determine whether it is possible to perform a non-destructive procedure in order to access device data. In this sense, one may request support from manufacturers and authorized service centers. Finally, there are methods of software access that, even though it depends on the handset model and Android version, are usually the easiest way and can be applied in the forensic analyst's own test environment.

To access the system, the analyst must do it the least intrusive manner possible in order to avoid compromising the evidence. If the password or the tactile pattern has been obtained when the device was seized, those should be readily tested. Alternatively, one may use the technique to find the tactile pattern by means of examining the smudge left on the device screen (Aviv, A. J. et al, 2010), before attempting any other way to bypass access control, preventing screen contamination.

If the analyst does not succeed, he should check if the Android is configured to accept USB debugging connections using a tool available in the SDK, the ADB. If he succeeds, he attempts to obtain "super user" access credentials to resume the acquisition process, the same way that it would be performed in cases where the mobile device was not locked, because with such permissions, one can get all the stored data in the device, as described previously.

Even when there is no "super user" access to the handset, it is still possible to install applications through the ADB tool to overcome the access control system. The technique described by Thomas Cannon (Cannon, T., 2011) is to install the "Screen Lock Bypass" application, available at the Android Market. In this technique, one needs that the Google account password be saved in the Android device, as well as Internet access be enabled, which is considered inadvisable. In this sense, it is recommended that the application is downloaded from another Android device and then installed via ADB on the examined mobile device. Thus, it is possible to perform the screen unlock using Cannon's technique without the need of having the device's Google account password or connecting it to the web. Figure xx shows Cannon's application installation, as well as its activation, which depends on the installation of any other application, to perform access control unlocking.



```
C:\android-sdk\platform-tools>adb -s 040140611301E014 shell
$ su -
su -
Permission denied
$ exit
...
C:\android-sdk\platform-tools>adb -s 040140611301E014 install screenlockbypass.apk
224 KB/s (22797 bytes in 0.100s)
        pkg: /data/local/tmp/screenlockbypass.apk
Success
C:\android-sdk\platform-tools>adb -s 040140611301E014 install AndroidForensics.apk
716 KB/s (31558 bytes in 0.046s)
        pkg: /data/local/tmp/AndroidForensics.apk
Success
```

Figure 44 Connection via ADB, root access check and application installation in order to ignore access control.

In situations where it is not possible to bypass the authentication system or USB debugging access is disabled, it is left to the analyst to copy the data contained in the removable memory card that may be installed on the handset. In those situations, it is very important to report the impossibility to access the device with the used procedures. In addition, if there is another technique that may be applied, be it more invasive or complex, that fact should be informed to whom requested the exams. Consequently, the implications of applying such techniques should be discussed, considering the risks to the given situation, such as permanent damages to the examined Smartphone.

Acquisition documentation

It is recommended that all the techniques and procedures used by the analyst be documented, in order to facilitate the examination and analysis of extracted data. Regardless of the path followed by the expert in the workflow illustrated in Figure xx, the process should be recorded, enabling auditability and reliability of the procedures performed by the expert analyst.
The analyst should be careful to register the hash codes of data generated and extracted during the acquisition process, as well as state in his report any caveats that he considers important to carry out the examination and analysis stage, like an e-mail or SMS received before the Smartphone have been isolated from telecommunication networks or even the existence of applications that contain information stored in servers on the Internet, such as cloud computing.
The forensic expert, while executing his activities, should consider that the better reported the acquisition process, the more trust will be given to the examination results. The simple condition of the processes be well documented is the first step to conduct an impartial, clear and objective data analysis.

4.2) Methodology of "Data Acquisition"

Overview

“Data acquisition” or sometimes simply “carving” is the process of extracting a collection of data from a larger data set. Data acquisition techniques frequently occur during a digital investigation when the unallocated file system space is analyzed to extract files. The files are “carved” from the unallocated space using file type-specific header and footer values.

File system structures are not used during the process. “File carving” is a powerful technique for recovering files and fragments of files when directory entries are corrupt or missing. The block of data is searched block by block for residual data matching the file type-specific header and footer values. Acquisition is also especially useful in criminal cases where the use of carving techniques can recover evidence. In certain cases related to child pornography, law enforcement agents are often able to recover more images from the suspect’s hard disks by using carving techniques. Another example is the hard disks and removable storage media US Navy Seals took from Osama Bin Laden’s campus during their raid. Forensic experts used file carving techniques to squeeze every bit of information out of this media.

As long as data is not overwritten or wiped, deleted data on all storage devices can be restored using carving techniques, including multifunctional devices and even mobile phones. Depending on the conditions, it is even possible to restore data from formatted disks. With the exhaustive measures of drives since 2006, there is a big chance that the data is not overwritten. For example, let's say we have a two-terabyte drive, and we delete a document from that drive. The disk space reserved for that document will be marked "available," but it could really take a long time before this address space on the disk is overwritten. There were forensic cases where we discovered files stored on the disk years ago.

Kinds of retrieved data

For people working in forensics, or interested in forensics, mobile phones are also very interesting sources of data. As with file systems, when we delete a file, it is only permanently deleted when it is overwritten by other data.

For mobile phones, it's the same. If an SMS message is deleted, it will still be in the flash memory of the phone or on the removable storage media until that memory space is overwritten. To be more specific, mobile phones use a type of solid state, nonvolatile memory known as flash memory to store SMS, call records, pictures, videos, and more. There are two types of flash memory that are commonly used: NOR and NAND. The NOR memory is used as the code storage media. Examples for items stored in the NOR memory are the phone's OS and default applications. The NAND memory is used for storing user data such as pictures and music.

Recovering data from a mobile phone is different. All phone models have an operating system: Microsoft Windows CE, Symbian, Android, and Mac OS X. These operating systems also store their files in the memory of the phone. Samsung, for example, makes use of the FAT file system. Every mobile phone vendor has its own way of storing data into the phone memory. Some vendors store the IMSI code (subscriber identification) in a certain field in the right order, but other vendors use "reverse nibbling" to store this code in the phone memory. We need to swap the individual nibbles of a byte to proper decode the data. For instance, 12h becomes 21h.

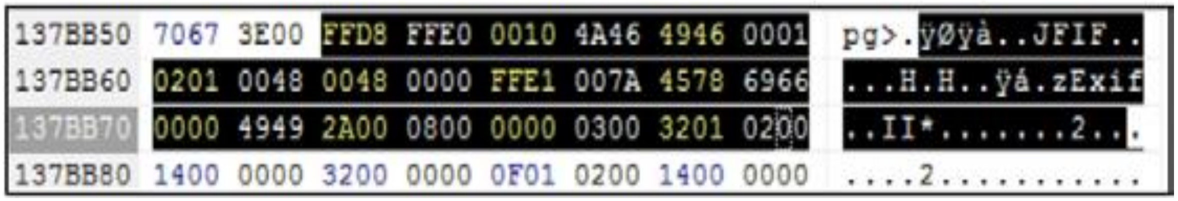
But how is it possible to recover data from a mobile phone? We need to understand the principles behind how the data is being stored on the mobile phone. Photos and music are usually stored on the onboard memory card. Once the card is available, data can be easily carved using a card reader and Photorec. For phone flash memory, a different procedure is required. For example, the content of an SMS message is compressed by the PDU format from eight ASCII characters into seven bytes. Alphabets may differ, and there are several encoding alternatives when displaying an SMS message.

There is no standard solution for recovering data from the flash memory of mobile phones. For computers, though, images of the disk and memory can be made by using the tool dd. For mobile phones, a flasher is leveraged to dump the physical file system of a mobile unit. An example of a flasher device is CellBrite's UFED Ultimate, used by many law enforcement and forensic investigators.

A hex dump is a snapshot of the entire contents of a handset's memory. Forensic examiners need to grab this data, preserve it and analyze it in the hope of finding information hidden from view and/or deleted data. Most of mobile phone forensic examination applications are a progression of backup software that concentrates on the user's data. Some of the applications have the functionality to decode the stored data, but many of them do not support the recovery of deleted items.

To overcome this problem, manual investigation of the dump seems to be the only solution. Mobile phones could contain file types like JPEG, MP3, MPEG, MOV, and others. Before manually searching, you need to define the file structure of each file type. For example, if we want to search for JPEG files in a dump from a cell phone, you could use the header and footer characteristics for JPEG as discussed above. These values are 0xFF D8 for the header and 0x FF D9 for the footer.

Open the dump in our favorite hex editor, and start searching for the string FF D8:



```
137BB50  7067 3E00 FFD8 FFE0 0010 4A46 4946 0001   pg>.ÿØÿà..JFIF..
137BB60  0201 0048 0048 0000 FFE1 007A 4578 6966   ...H.H..ÿá.zExif
137BB70  0000 4949 2A00 0800 0000 0300 3201 0200   ..II*.......2...
137BB80  1400 0000 3200 0000 0F01 0200 1400 0000   ....2...........
```

Figure 45 Examining a raw phone dump for JPEG.

After discovering a possible JPEG file, mark this beginning position and start looking for the values of the footer. When we have discovered the footer, select the block of data and save it to disk as a JPEG file. While opening it in a file viewer, the following image appears:

Figure 46 Acquainted image from cell phone dump.

In this case, we were lucky, the header and footer belonged to the same JPEG file. Often you will notice that the images you retrieve by hand are incomplete. As stated previously, the way mobile phones store their data depends on the manufacturer or operating system, so the files you are looking for could be heavily fragmented.

Design Phases

In order to analyze and evaluate any forensic tool a sufficient amount of data need to be collected. To make it in a structured way the research design has divided into four stages. The first stage is to test the capability of the Android phone. Android phones maintain a high level of security both kernel level and application level. Though this is a bad news for a forensic examiner there are number ways the data can be stored and extracted. Apart from secure nature Android phones also store enormous amount of information related to system information, user data, application data, audio, video files etc. Evaluating what information can be stored on the internal memory, external memory or in the lightweight databases (like SQLite). Android phones are highly volatile when retrieving this information especially from the memory.

The second step of the research involves the extraction tools testing. Extraction tools selected for testing are the industry leading tools. While acquiring some of these tools involves huge cost the test has been done with the lite version or open source version of the same tools with some of the features are not available. A testing environment has been created with Android Software development kit. Two different models of phones from the same manufacturer have been selected for testing. This is to make sure all the forensic tools should supported by the extraction tools. Not all forensic tools support the entire majority of Android phones from many different hardware makers. The forensic artifacts are counted and benchmarked. Each forensic tool has tested against this benchmark data.

The next step consists mainly to compare the extraction tools. The comparison has based on the baseline artifacts that manually counted from the phones. The comparison is also done with the cost involved in procuring the software if a commercial based tool is used against the free open source. While the most open source tools are require expert examiner to figure out what could be done in the tool configuration when initiating a forensic analysis. The ranking of tools are quiet hard as there is no such benchmark can enforce on the tools that employ different methodology to extract data. Some of the tools extract the data from the SD card or eMMC card while others don’t extract any of the data from the image. The forensic analysis of this external memory card doesn’t face much problem for a forensic investigator as these cards are partitioned with either with FAT or FAT32. The techniques employ to analyze the external memories are the same as like in the computer hard disk. To ensure the accuracy of the tool testing a manual method of extraction of phone has been conducted in this phase.

Final step involves the graphical representation of the data carved out from the mobile phones by the forensic tools. Each tool uses different methodologies to get the extraction done. The graphical representation can be the best way to evaluate and compare these extraction tools as the data stored in the Android phones are enormous and it require more time to analyze and evaluate if done with the other way.

Data Collection

Data collection is an important part of the research. The testing and evaluation of the tools produces enormous amount of data. These data is important to analyze the forensic evidences. There are three sources of data for this research. The first phase of the project returns data which is of probative value to investigators and that needs to be analyzed. These data provide the baseline for testing each of the extraction tools. The actual forensic evidence will be collected from each of the phones so that this can be compared with the artifacts that have counted manually. The third and final sources of data are the evaluation outcomes with respect to the baseline data. These data are the actual data for the purpose of the visual representation of evaluation and comparison.

The data collection and analysis has been done with asking main question and sub question. The main question is: what is the capability of Android devices for a device analysis. Based on the several questions a data map has been created. The data map clearly identifies how data can be identified, collected and processed.

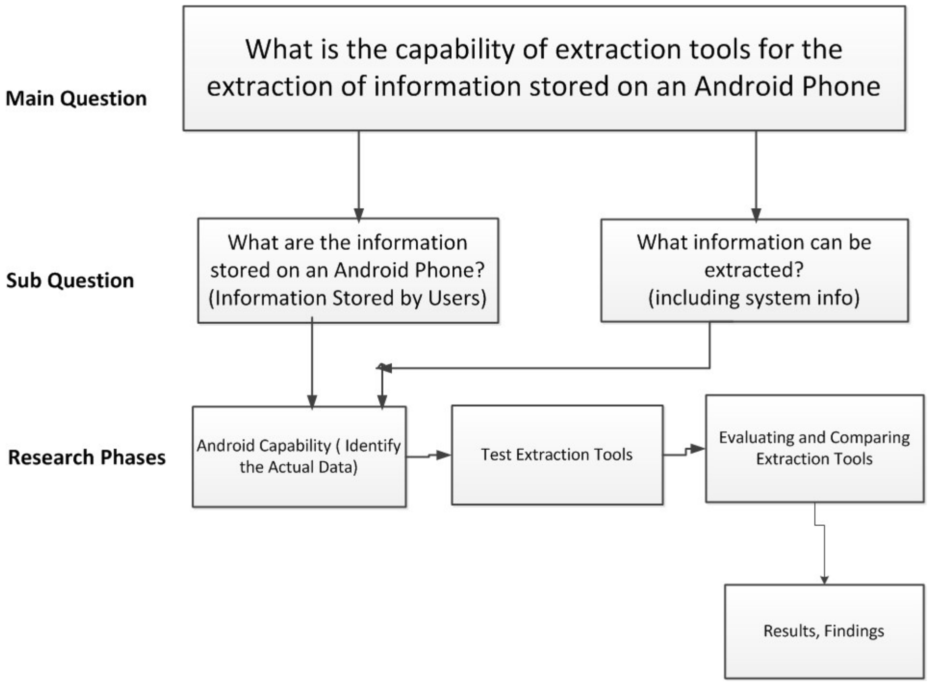

Main Question
What is the capability of extraction tools for the extraction of information stored on an Android Phone
Sub Question
What are the information stored on an Android Phone? (Information Stored by Users)
What information can be extracted? (including system info)
Research Phases
Android Capability (Identify the Actual Data)
Test Extraction Tools
Evaluating and Comparing Extraction Tools
Results, Findings

Figure 47 Data map

The final stage of the data analysis is the data presentation. The results of the data during each stage of the tool testing will be presented with countless data. A clear visual representation of the result has been converted to get the comparative results. The results are graphed in each stage of the testing.

4.2.1) Extraction tools

Mobile forensics is relatively new area of digital forensics and the software and tools required to extract data from the mobile phones are still at the nascent stage. Extraction tools can be hardware or software based depending on how the data is extracted from mobile device. There are a number of extraction tools available in the market today and newer tools are emerging with some innovative ideas It is found that almost all tools are commercial ones and few number of open source tools. However, procuring these tools found to be very difficult given the privacy and security issues and the cost involved. As physical extraction was not considered in this project so the leading tool like XRY, Cellebrite UFED, FTK, Paraben Cell Seizure, Neutrino, Oxygen Forensic Suite and Mobiledit, LiME, Frost ADB, Custom Recovery ROM and Recovery Partition are omitted. Other leading tools like Access Data Mobile phone Examiner Plus (MPE+) and via Forensics via Extract found to be difficult to procure.

4.2.1.1) Method of extraction

Mobile phones lack the traditional hard drives like in computers making it hard for the examiner to extract the data. In most cases it is up to the examiner's discretion whether to take a note of the modification happened. Any alteration of target device has to be reported and kept aside for future reference with a detailed explanation why and how the alteration has happened.
Brothers (2009) outlined five phases of extraction from the mobile devices (Figure xxx). As shown in the figure the levels are divided two ways and are physical methods which are more forensically sound extraction methods but high level of technical competency required (Brothers, 2009). The bottom most layers are the manual methods which involves manually reviewing a mobile device by simple straight through process. Manual method of extraction is very simple to perform and almost all devices can be analyzed this method. The problem with method is there are chances that the examiner may miss out some critical area of evidence like deleted items. This could be serious implication while submitting the evidence to the court. This method is only suitable in situations where integrity of data is not that much important and a very limited time to produce the evidence.

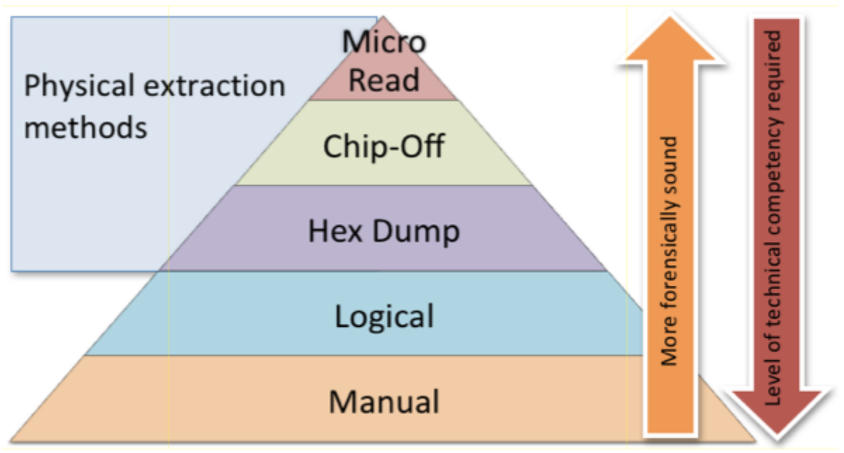

Physical extraction methods
Micro Read
Chip-Off
Hex Dump
Logical
Manual
More forensically sound
Level of technical competency required

Figure 48 Forensic Tool Analysis Pyramid – Source: Brothers, (2009)

The next extraction method is logical. This is mostly recommended method of data extraction. This technique involves copying a small Android Forensic application to the device then removing from the device. Via-Extract is one such application from the company called Via-Forensics, it extract following information:

- Browser history
- Call Logs
- Contact Method
- External Image Media (meta data)
- External Image Thumbnail Media (meta data)
- External Media, Audio, and Misc. (metadata)
- External Videos (metadata)
- MMS
- Organizations
- People
- SMS
- List of all applications installed and version
- Contacts Extensions
- Contacts Groups
- Contacts Phones
- Contacts Settings (Hoog, Introduction to Android forensic, 2010).

Logical extraction is quick to perform and not require a high level of technical expertise. The method is repeatable however still can't be considered forensically sound as changes may be made to the data on the device during the coping process (Westman, 2009).
The next three layers are physical extraction methods. Physical extractions are more into technical in nature but provide sound forensic results. The first physical layer is hex dump. Hex dumping involves uploading an altered boot loader to the device and boot from the newly loaded bootloader. This method can create a forensically sound image. Hex dumping of a mobile device is similar to booting a computer from a boot CD and acquiring image of the hard disk while computer running. Hex dumping requires a high level of technical expertise as this is mainly interacting with the code level access of the operating system (Brothers, 2009).
The next layer is 'chip-off' and this technique is to remove NAND flash chips physically and examined externally. This technique mainly used when the device is damaged and in some cased to override the passcode-protected devices. This process is quite a destructive process and attempted as a last resort when nothing is worked mainly due to damage of the NAND chip during this process (Hoog, Android forensic techniques, 2011).

The final extraction method is micro read that requires the most technical expertise to perform. The micro-read method uses an electron microscope to view the state of the memory on the device. But this method involves huge costs and not used a standard method. (Knight, 2010).

4.2.1.2) Evaluation criteria

Forensic examiners use the term 'Forensically Sound' when referring the forensic investigation. The main purpose of keeping the forensic evidences as forensically sound so that the data collected from the devices should not lose its evidential value when using in the court. One of the main requirements of a forensic tool is to produce the evidences that are forensically sound. There are several criteria for evaluating the forensic tools. However these criteria may change depends how an examiner conducts the testing. Reliability and Completeness is the two most important qualities that must be checked when evaluating the forensic tools. If both of these in question the evidential weightage will be in question and eventually less chances that these can be proved in the court.

4.2.1.3) Testing and validation

The outlined simple methodology to perform the analysis of the mobile devices. The steps are illustrated in Figure 49:

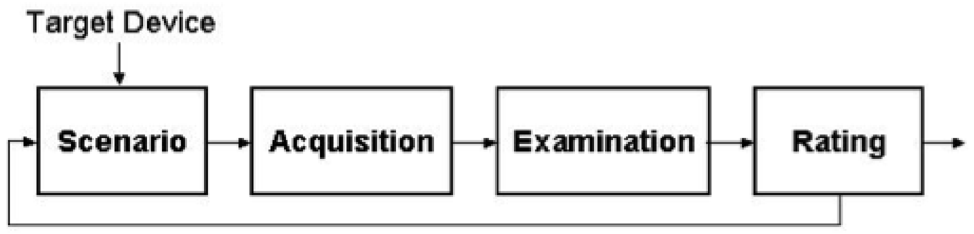


Figure 49 Tool Assessment (Ayers, et al, 2007, p.16)

Ayers, et al (2007) followed some simple methods by acquiring a set of target devices and then followed a set of prescribed activities like placing and receiving calls was performed on each phone. After that the contents of the phone and associated SIM were acquired using an available tool and examined to determine whether the evidence of the activity is recovered as expected. Finally a report has been taken (Rick Ayers, 2007).

4.2.2) Mobile forensics applications

There are a number of anti-theft applications (anti-theft applications) protection which may facilitate the identification of the mobile device is stolen or lost. At the same time, however, the installation of such applications offers an unlimited number of remote access capabilities to the mobile device, where the installation of a malicious, can act as spyware application.

4.2.2.1) Where's My Droid[18]

Allows us to specify a keyword which when sent to mobile with SMS, that responds to the position based on the GPS, so you can locate it. Even we can send a keyword via SMS to the phone to increase the volume if it was lowered or on silent.

Where's My Droid
Lose it. Track it. Find it.
The Original "Find Your Phone" App on Android

Figure 50 WheresMyDroid

4.2.2.2) SeekDroid[19]

It offers a variety of features to help you track your lost or stolen phone. You can use the website to locate your device via GPS, activate the alarm on the phone to lock or even delete the data from the card SD.

[18] http://wheresmydroid.com/
[19] https://seekdroid.com/v2/

SEEKDROID
SEEKDROID TRACKER
Kind of like being the CIA...except more cell phones, less Communists.

Figure 51 SeekDroid

4.2.2.3) Prey Anti-Theft[20]

It allows us to track the GPS location of the mobile phone or tablet by sending a message (SMS). We can track it via the web interface, or enable remote locking device. The application works even if we change the card SIM. Also the software can be used to track a laptop.

Figure 52 Prey Anti-Theft

[20] http://preyproject.com/

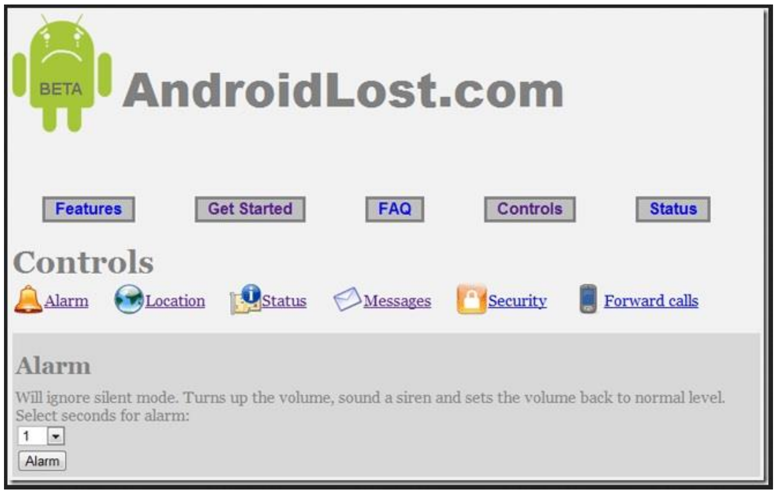
University of Piraeus

4.2.2.4) Androidlost[21]

Application locates the mobile in case of theft and control of mobile remotely via the web interface of the application. It can work without being connected to the internet, after the user sends commands to identify (GPS) of the mobile via SMS.

Below are summarized the main features of the application:

- Read SMS via email (use the account to which you activated the mobile),
- Send SMS to mobile through the site,
- Play loud sound (if you left it somewhere in the house),
- View location (either over the network or via GPS for greater accuracy),
- Locking mobile code,
- Erase personal data (SMS, contacts, settings) and the entire SD card



Will ignore silent mode. Turns up the volume, sound a siren and sets the volume back to normal level.
Select seconds for alarm:

Figure 53 AndroidLost

21 http://www.androidlost.com/

4.2.3) Mobile Forensics Tools for "Data Acquisition" processing.

4.2.3.1) XRY

XRY[22] is a the contents of a device in digital forensics product by Micro Systemation used to analyze and recover information from mobile devices such as mobile phones, smartphones, GPS navigation tools and tablet computers. It consists of a hardware device with which to connect phones to a PC and software to extract the data. It is designed to recover a forensic manner so that the contents of the data can be relied upon by the user. Typically it is used in civil/criminal investigations, intelligence operations, data compliance and electronic discovery cases. The software is available to law enforcement, military and intelligence agencies. It has become well known in the digital forensics community as one of their common tools for this type of work.

There are many more complex challenges when examining mobile phones in comparison to the forensic examination of normal computers. Many mobile phones have their own proprietary operating systems, which makes reverse engineering of such devices a very complex operation. The speed of the mobile device market also means that there are many more new devices being manufactured on a regular basis, so a mobile forensics tool must deal with all of these issues before being suitable for the task.

The XRY system allows for both logical examinations (direct communication with the device operating system) and also physical examinations (bypassing the operating system and dumping available memory). Whilst the logical recovery of data is generally better supported for more devices, physical examination offers the ability to recover more deleted information such as SMS text messages, images and call records etc. Because of the complexities of the topic, specialist training is usually recommended to operate the software.

The latest versions include support to recover data from smartphone apps such as the Android, iPhone and Blackberry devices. Data recovered by XRY has been used successfully in various court systems around the world.

XRY has been tested by a number of different government organizations as suitable for their needs and is now in worldwide use.

[22] http://en.wikipedia.org/wiki/XRY_(software)

Figure 54 XRY

4.2.3.2) Cellebrite UFED

The **Cellebrite 'Universal Forensic Extraction Device' (UFED)**[23] is a tool for mobile phone, Smartphone, and PDA forensics. As of September 2010 the UFED was compatible with over 2,500 mobile phones (including GSM, TDMA, CDMA, iDEN). The standard package containing several dozen phone cables. The UFED had an intergrated SIM reader, with Wireless connection options also being integrated, such as IR and Bluetooth.

The UFED also supports native Apple iPOD Touch, and Apple iPHONE extraction on both 2G and 3G versions, as well as iOS4. This is clientless, and via a physical cable, and works on jailbroken and non-jailbroken devices.

Subject data can be retrieved via logical extraction or via physical extraction (ie: hex dump). Moreover, all cable connectors from subject (source) side act as a write-blocker, being read only via the onboard hardware chipset. Extracted data includes basic handset data, the phonebook, SMS and MMS messages, SIM data, multimedia (e.g. images and videos stored on the phone), and time and date stamps.

[23] http://www.cellebrite.com/mobile-forensic-products

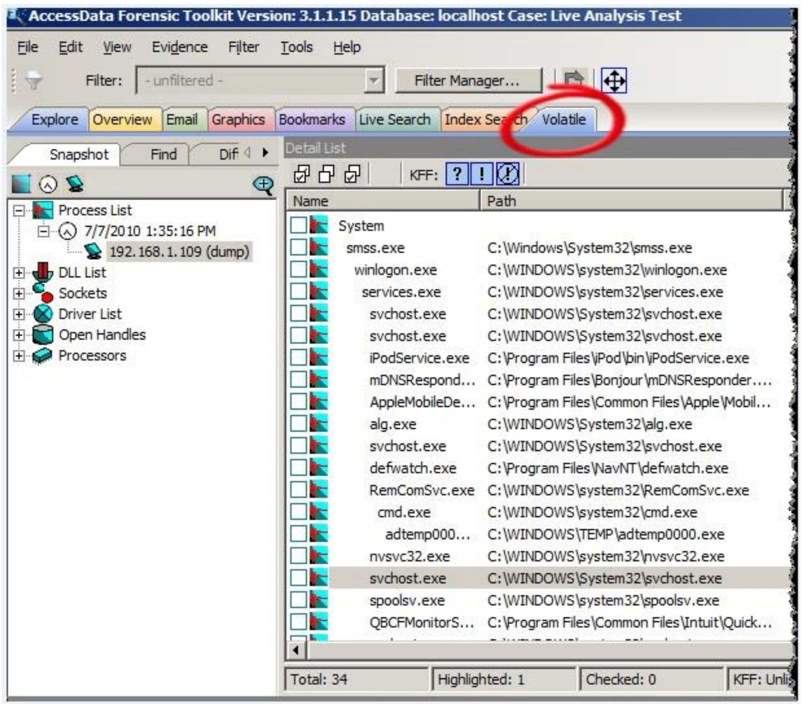
Figure 55 Cellebrite UFED

4.2.3.3) FTK

Forensic Toolkit[24], or FTK, is a computer forensics software made by AccessData. It scans a hard drive looking for various information. It can for example locate deleted emails and scan a disk for text strings to use them as a password dictionary to crack encryption.

The toolkit also includes a standalone disk imaging program called **FTK Imager**. The FTK Imager is a simple but concise tool. It saves an image of a hard disk in one file or in segments that may be later on reconstructed. It calculates MD5 hash values and confirms the integrity of the data before closing the files. The result is an image file(s) that can be saved in several formats including, DD raw.

[24] http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk


AccessData Forensic Toolkit Version: 3.1.1.15 Database: localhost Case: Live Analysis Test

Name	Path
System	
smss.exe	C:\Windows\System32\smss.exe
winlogon.exe	C:\WINDOWS\system32\winlogon.exe
services.exe	C:\WINDOWS\system32\services.exe
svchost.exe	C:\WINDOWS\System32\svchost.exe
svchost.exe	C:\WINDOWS\System32\svchost.exe
iPodService.exe	C:\Program Files\iPod\bin\iPodService.exe
mDNSRespond...	C:\Program Files\Bonjour\mDNSResponder....
AppleMobileDe...	C:\Program Files\Common Files\Apple\Mobil...
alg.exe	C:\WINDOWS\System32\alg.exe
svchost.exe	C:\WINDOWS\System32\svchost.exe
defwatch.exe	C:\Program Files\NavNT\defwatch.exe
RemComSvc.exe	C:\WINDOWS\system32\RemComSvc.exe
cmd.exe	C:\WINDOWS\system32\cmd.exe
adtemp000...	C:\WINDOWS\TEMP\adtemp0000.exe
nvsvc32.exe	C:\WINDOWS\system32\nvsvc32.exe
svchost.exe	C:\WINDOWS\System32\svchost.exe
spoolsv.exe	C:\WINDOWS\system32\spoolsv.exe
QBCFMonitorS...	C:\Program Files\Common Files\Intuit\Quick...

Total: 34 Highlighted: 1 Checked: 0 KFF: Unli

Figure 56 FTK version 3.1.1.15

}

4.2.3.4) Paraben Cell Seizure

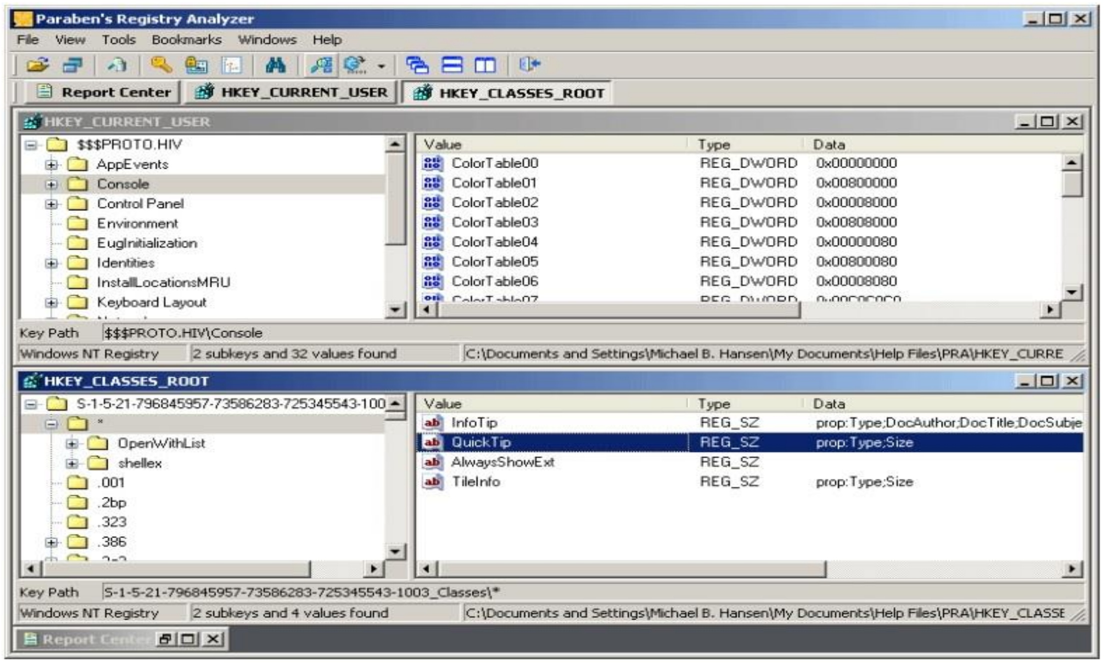
Paraben Cell Seizure[25] is a piece of software that serves the main purpose of collection and examining data pulled from various types of cell phones. It is part of Paraben's collection of forensics tools.

The main goal of Cell Seizure is to organize and report various types of files. Following the same guidelines as other Paraben products, Cell Seizure is able to generate comprehensive HTML reports of acquired data. Moreover the software is able to retrieve deleted files and check for file integrity.

One important advantage that separates Paraben's Cell Seizure from other similar products is the fact that it is designed not to change the data stored on the SIM card or cell phone. In other words, all of the data can be examined while keeping the process undetected. Most commercial or free software for cell phones is designed do not only view data but to upload data to cell phones. This is not a safe way to perform a forensic evaluation of cell phone data. In fact, even some forensic software warns of possible data loss. Cell Seizure does not allow data to be changed on the phone.

[25] https://www.paraben.com/device-seizure.html

A disadvantage of Cell Seizure is that it does not support all makes and models of cell phones. As with most mobile device forensics software, Cell Seizure is frequently updated with support for new devices as they proliferate in the consumer market. However, this application can acquire information from most models made by the following companies: Nokia, LG, Samsung, Siemens, Motorola, Sony-Ericcson, and can also acquire GSM SIM Cards. Another disadvantage would be that the format of acquired data can sometimes be confusing. The data is not organized nice and neat and given to the user in a way that they can easily understand what they are seeing. It is not the most user-friendly way of displaying the data that has been acquired.


Paraben's Registry Analyzer
File View Tools Bookmarks Windows Help
Report Center
HKEY_CURRENT_USER
HKEY_CLASSES_ROOT

HKEY_CURRENT_USER

```
$$$PROTO.HIV
  AppEvents
  Console
  Control Panel
  Environment
  EugInitialization
  Identities
  InstallLocationsMRU
  Keyboard Layout
```

Value	Type	Data
ColorTable00	REG_DWORD	0x00000000
ColorTable01	REG_DWORD	0x00800000
ColorTable02	REG_DWORD	0x00008000
ColorTable03	REG_DWORD	0x00808000
ColorTable04	REG_DWORD	0x00000080
ColorTable05	REG_DWORD	0x00800080
ColorTable06	REG_DWORD	0x00008080

Key Path $$$PROTO.HIV\Console

Windows NT Registry | 2 subkeys and 32 values found | C:\Documents and Settings\Michael B. Hansen\My Documents\Help Files\PRA\HKEY_CURRE

HKEY_CLASSES_ROOT

```
S-1-5-21-796845957-73586283-725345543-100
  *
    OpenWithList
    shellex
  .001
  .2bp
  .323
  .386
```

Value	Type	Data
InfoTip	REG_SZ	prop:Type;DocAuthor;DocTitle;DocSubje
QuickTip	REG_SZ	prop:Type;Size
AlwaysShowExt	REG_SZ	
TileInfo	REG_SZ	prop:Type;Size

Key Path S-1-5-21-796845957-73586283-725345543-1003_Classes*

Windows NT Registry | 2 subkeys and 4 values found | C:\Documents and Settings\Michael B. Hansen\My Documents\Help Files\PRA\HKEY_CLASSE

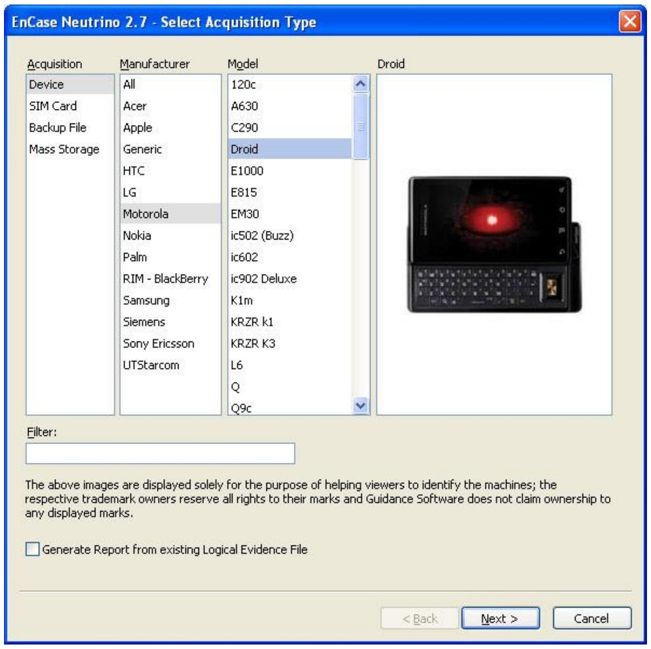
Figure 57 Paraben Cell Seizure

4.2.3.5) Neutrino

Neutrino[26] is the only mobile device acquisition tool that integrates with EnCase® v6, allowing you to analyze both mobile devices and computer evidence at the same time. Because EnCase v6 software delivers analysis of data acquired by Neutrino, you can analyze and document both mobile device data and computer data simultaneously to correlate all evidence during an investigation.

[26] http://www.mobileforensicscentral.com/mfc/products/neutrino.asp?pg=d&pid=&prid=329&return=undefined

The Neutrino WaveShield signal-blocking bag has been tested directly under cell towers to ensure reliability. It is not uncommon for an investigator to need the positions for the last cell towers, or longitude and latitude for their last contact, prior to the analysis. There are many wireless signal blocking technologies available, but many fail to dependably block wireless signals to the device being examined. Guidance Software has conducted extensive research and testing to create a reliable signal-blocking bag that allows for the safe acquisition of mobile device data, even within close proximity to cell towers.



The above images are displayed solely for the purpose of helping viewers to identify the machines; the respective trademark owners reserve all rights to their marks and Guidance Software does not claim ownership to any displayed marks.

☐ Generate Report from existing Logical Evidence File

Figure 58 Neutrino

4.2.3.6) Oxygen forensic suite

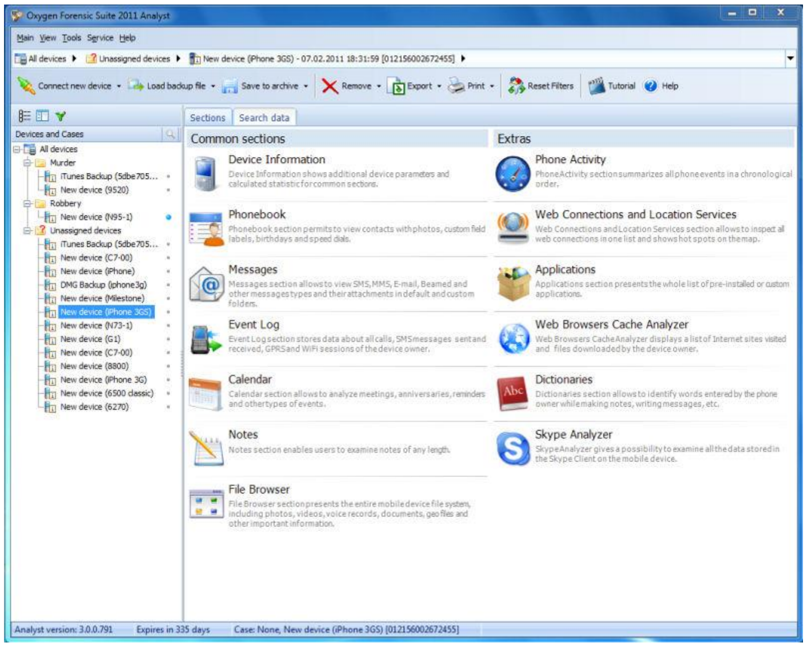
Oxygen Forensic Suite[27] is mobile forensic software that goes beyond standard logical analysis of cell phones, Smartphones and PDAs. Using advanced proprietary protocols permits Oxygen Forensic Suite to extract much more data than usually extracted by logical forensic tools, especially for Smartphones. Oxygen Forensic Suite – Smart Forensics for Smart Phones. Using low-level protocols allows the program to extract: phone basic information and SIM-card data, contacts list, caller groups, speed dials, missed/outgoing/incoming calls, standard SMS/MMS/E-mail folders, custom SMS/MMS/E-mail folders, deleted SMS messages (with some restrictions), SMS Center timestamps, calendar events schedule, tasks, text notes, photos, videos, sounds, LifeBlog data (all main phone events with their geographical coordinates), Java applications, file system from phone memory and flash card, GPRS and Wi-Fi activity, voice records and much more.

[27] http://www.oxygen-forensic.com/en/

The list of supported features depends on a certain phone model. Oxygen Forensic Suite combines the simplicity and cheapness of logical forensic software and the extracted information completeness of physical tools. Device Connection Wizard helps you to connect a phone in several mouse clicks. Data Extraction Wizard works in unattended mode and downloads all the available device information in just few minutes. After the download process is completed, you can either select the automatic forensic report generation function or use convenient program interface to analyze, filter and search for the extracted data.

More than 2200 mobile device models are supported. And the list is rapidly growing! Oxygen Forensic Suite extracts data from Nokia, iPhone, Sony Ericsson, Samsung, Motorola, Blackberry, Panasonic, Siemens, HTC, HP, E-Ten, Gigabyte, i-Mate, Vertu and many other mobile phones.

The future of cell phone market is Smartphones. Oxygen Forensic Suite supports Symbian OS, Nokia S60, Sony Ericsson UIQ, Windows Mobile 5/6 (without using ActiveSync!), Blackberry, Android and Apple Smartphones.



Common sections

Device Information
Device Information shows additional device parameters and calculated statistic for common sections.

Phonebook
Phonebook section permits to view contacts with photos, custom field labels, birthdays and speed dials.

Messages
Messages section allows to view SMS, MMS, E-mail, Beamed and other messages types and their attachments in default and custom folders.

Event Log
Event Log section stores data about all calls, SMS messages sent and received, GPRS and WiFi sessions of the device owner.

Calendar
Calendar section allows to analyze meetings, anniversaries, reminders and other types of events.

Notes
Notes section enables users to examine notes of any length.

File Browser
File Browser section presents the entire mobile device file system, including photos, videos, voice records, documents, geo files and other important information.

Extras

Phone Activity
Phone Activity section summarizes all phone events in a chronological order.

Web Connections and Location Services
Web Connections and Location Services section allows to inspect all web connections in one list and shows hot spots on the map.

Applications
Applications section presents the whole list of pre-installed or custom applications.

Web Browsers Cache Analyzer
Web Browsers Cache Analyzer displays a list of Internet sites visited and files downloaded by the device owner.

Dictionaries
Dictionaries section allows to identify words entered by the phone owner while making notes, writing messages, etc.

Skype Analyzer
Skype Analyzer gives a possibility to examine all the data stored in the Skype Client on the mobile device.

Figure 59 Oxygen Forensic Suite

Memdump

This program dumps system memory to the standard output stream, skipping over holes in memory maps. By default, the program dumps the contents of physical memory. This sofware is distributed under the IBM Public License.

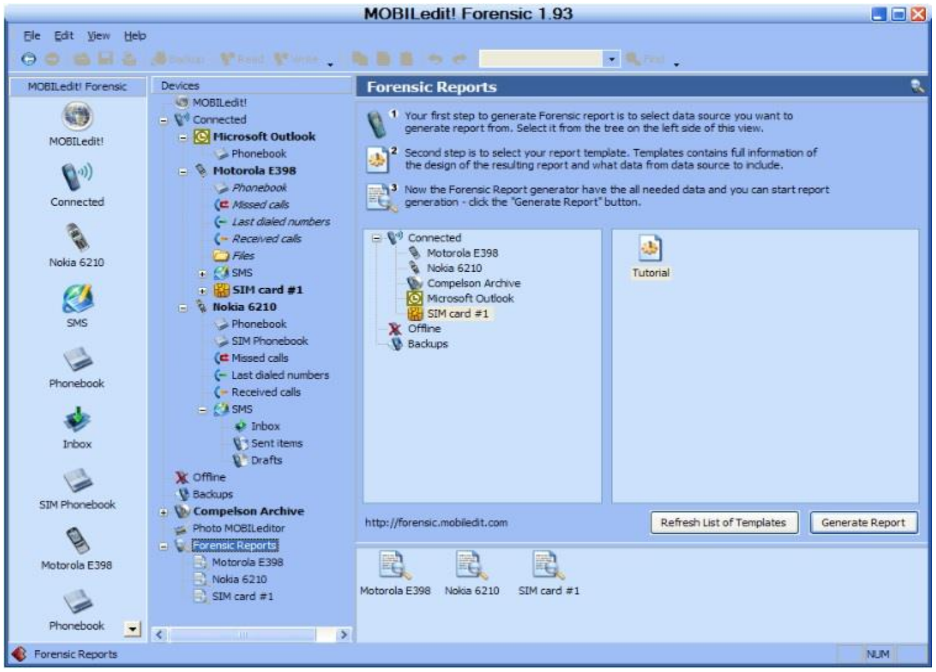
AFlogical

Via Forensic AFlogical OSE is an open source tool to extract the data. This is lightweight software purely used in the command line. This tool utilizes the Android adb feature to communicate with a computer.

4.2.3.7) Mobiledit Forensic

MOBILedit[28] Forensic tool allows examiners to acquire logically, search and examine the mobile phone devices. This tool uses multiple connectivity mechanism than other similar tools especially the wireless connectivity. The software is well enough to acquire the phone system information and other contacts and messages lists.

Especially with MOBILedit Forensic we can view, search or retrieve all data from a phone with only a few clicks. This data includes call history, phonebook, text messages, multimedia messages, files, calendars, notes, reminders and application data such as Skype, Dropbox, Evernote, etc. It will also retrieve all phone information such as IMEI, operating systems, firmware including SIM details (IMSI), ICCID and location area information. Where possible MOBILedit Forensic is also able to retrieve deleted data from phones and bypass the passcode, PIN and phone backup encryption.



1 Your first step to generate Forensic report is to select data source you want to generate report from. Select it from the tree on the left side of this view.

2 Second step is to select your report template. Templates contains full information of the design of the resulting report and what data from data source to include.

3 Now the Forensic Report generator have the all needed data and you can start report generation - click the "Generate Report" button.

Figure 60 MOBILedit Forensic

[28] http://www.mobiledit.com/

4.2.3.8) Linux Memory Extractor Tool (LIME forensics).

LiME[29] (Linux Memory Extractor, formerly DMD) is a tool used to acquire complete memory captures from Android. It dumps all volatile data into a single file either in SD card or via the network. It minimizes the interaction between user data and kernel space, which allows it to produce memory dumps that are more forensically sound than those of other tools designed for Linux memory acquisition.

There are problems when traditional tools are trying to acquire an Android memory‘s image. It is difficult to obtain the starting offset to copy and does not match exactly the one given by the read operation. There is also a problem distinguishing and omitting the physical address space that is not RAM and usually mapped to other devices.
LiME solves those problems. It parses the kernel to learn the physical memory address ranges of system RAM and performs a translation form physical to virtual address for each page of memory to be easily read. Then, it reads all pages in each range and writes them to the aforementioned single file.

LiME is a Loadable Kernel Module (LKM), which is cross compiled in order to use it with an Android OS. It needs all the Android utilities (NDK, SDK), the kernel source of the already rooted device. First, a makefile has to be created in order to cross compile the kernel module, as LiME is device specific.

4.2.3.9) Forensics Recovery of Scrambled Telephones (FROST)

Forensics on Smartphones has become crucial in present day, as a person‘s life is stored in there. Main problem is the scrambled phones that the OS uses scrambled user partitions to protect sensitive user information. This is the main feature of the Android 4.0 OS, which has become the most popular Smartphone OS out there and thus the main concern. This feature prevents IT forensics and law enforcement to access the data once the power is cut off and the only remaining chance of reading those stored data is bruteforce.

A solution to this problem is **FROST**[30] (Forensics Recovery of Scrambled Telephones). This is a toolset that supports the forensics recovery of scrambled phones, as its acronym declares. It utilizes cold boot attacks in order to preserve the volatile data remaining on the RAM and find useful information.

[29] https://lime-forensics.googlecode.com/files/LiME_Documentation_1.1.pdf
[30] http://www.cisa.umbc.edu/papers/DFRWS2013_Dykstra_FROST.pdf

Cold Boot Attacks

This type of attack belongs to the group of side channel attacks, which are attacks that are based on information gained from the physical implementation of a cryptosystem. The main goal of this attack is to retrieve cryptographic keys from a running OS. Cold booted refers to when the device is cycled off and then on without letting an OS run the proper shutting down scripts. It is based on the characteristic of the RAM to preserve data for a limited period after power has been removed from the device, either by removing a device‘s battery or unplugging it. Unlike the contemporary perception, cryptographic keys are not protected from attackers with physical access (Princeton, 2008).

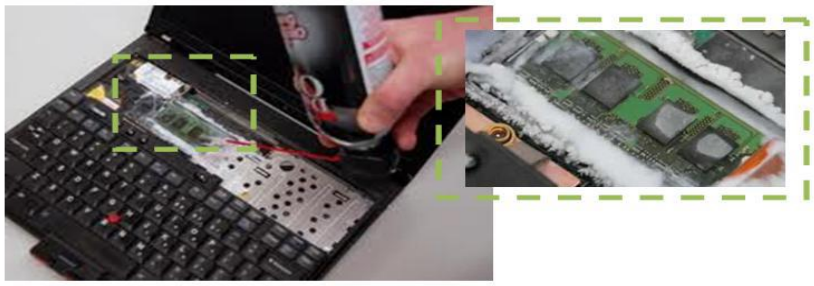


Figure 61 Cold Boot Attack by FROST

The attack has been demonstrated to be effective against full disk encryption schemes of various operating systems. With certain memory modules, the time window for an attack can be extended by cooling either the RAM or, when not possible, the whole device.

Results of shutting down a machine are depicted in the degradation of this image below, in accordance to the time passing. It is a bitmap image that is loaded in the memory on a machine. Then the machine becomes unplugged and the memory is checked to find out how quickly the volatility comes into play.

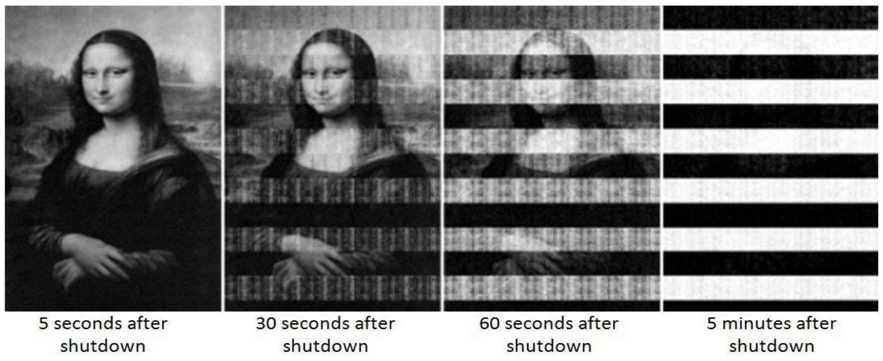


Figure 62 Degradation of the image at various times

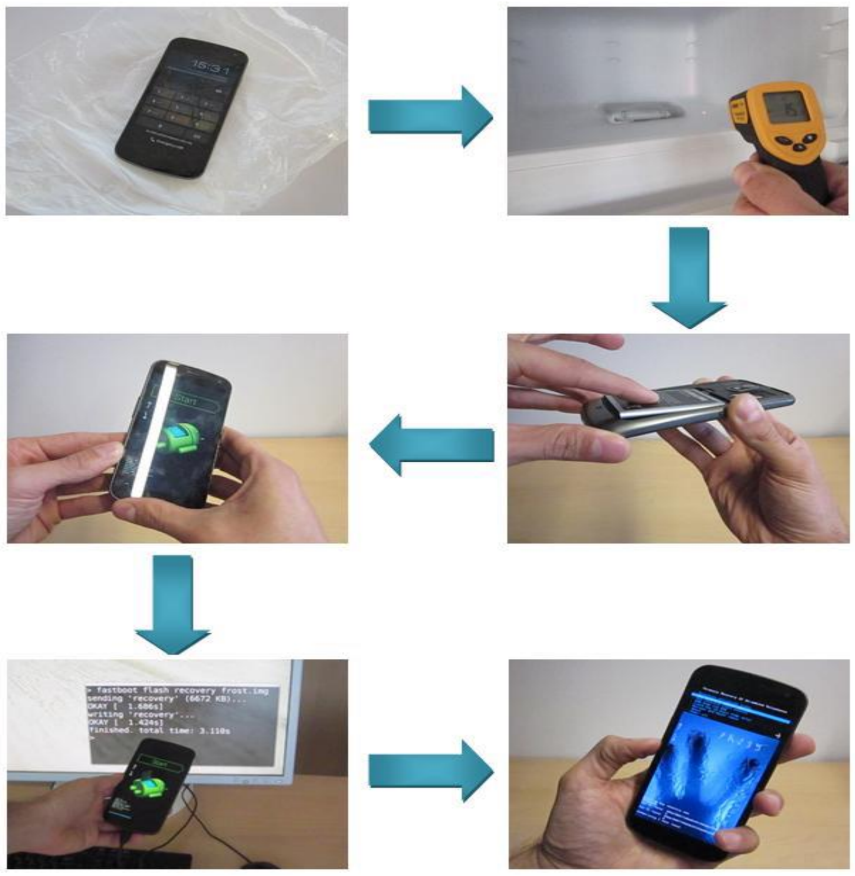
The use of cold boot is obvious, as the need to prolong the life of data after shutting down is essential.

Practical guide of FROST

FROST uses cold boot attack in order to discover keys and attain the most part of the memory. Let‘s assume that a forensics analyst has physical access to a device that is scrambled and locked with a PIN number. The analyst wants to gain access to its data but does not know the PIN. He freezes the phone inside a -15°C for approximately an hour to increase the success of the cold boot attack, decreasing the volatility property of the RAM.

After the _freezing‘ hour, the telephone is checked if it still works. Then, he reboots the device by removing the battery without shutting it down properly. The procedure of removing the battery has to be instantaneous (~500 ms) so as not to have data loss. Afterwards, he boots the phone in fastboot mode using a phone specific combination (usually holding simultaneously power button and volume up or/and down). The phone is connected to a Linux PC, that has all the fastboot utilities installed and then he flashes the frost image file () into the recovery segment of the phone. After the flash, he chooses recovery mode and FROST is initialized with all the possible options to choose from.

Practical Overview



```
> fastboot flash recovery frost.img
sending 'recovery' (6672 KB)...
OKAY [  1.686s]
writing 'recovery'...
OKAY [  1.424s]
finished. total time: 3.110s
>
```

Figure 63 Phases of Cold Boot Attacking

Graphical user interface of the Recovery Image

The recovery image of FROST provides to the users some options, as depicted in the image below.

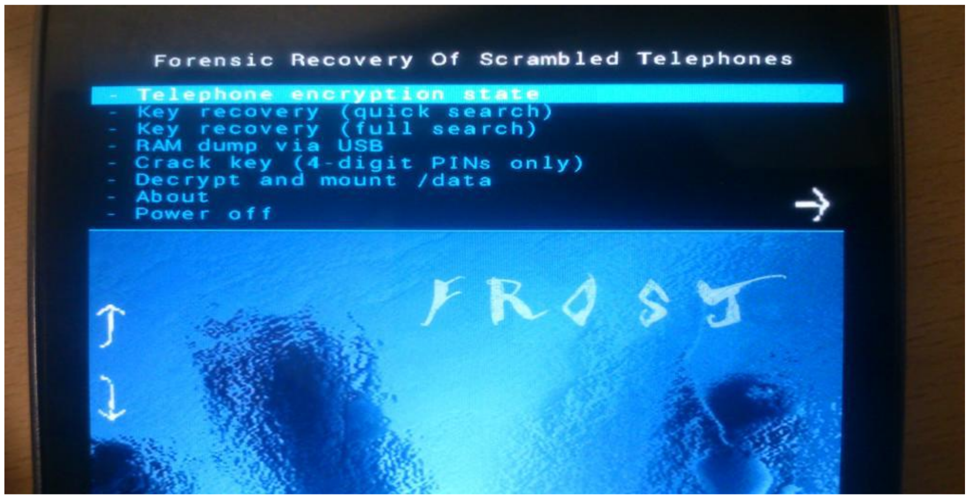

Forensic Recovery Of Scrambled Telephones
- Telephone encryption state
- Key recovery (quick search)
- Key recovery (full search)
- RAM dump via USB
- Crack key (4-digit PINs only)
- Decrypt and mount /data
- About
- Power off
FROST

Figure 64 Graphical interface of FROST.

Telephone encryption state: To check the encryption state of the phone, simply tries to mount the user data partition and check whether that fails.

Key recovery (quick search): Induces a quick search for AES keys. Internally, the Frost LKM is loaded and its output is displayed to the user.

Key recovery (full search): Induces a full search for AES keys. Also here, the Frost LKM is loaded.

RAM dump via USB: To load full memory dumps to the PC, it makes use of the LiME module. Similar to the Frost LKM, LiME parses a kernel structure to learn physical memory addresses. It then maps each physical page that is in system RAM, i.e., that does not belong to an I/O device, to a virtual address in kernel space. Each mapped page is transferred over a TCP socket to the computer via USB.

Crack 4-digit PINs: Performs brute force attacks against weak PINs.

Decrypt and mount /data: Decrypts the user partition with recently recovered keys. To decrypt the user data partition, we integrated a statically linked

ARM binary of the dmsetup utility: This option becomes available only if one of the key recovery methods or the brute force approach were successful, i.e., only after the decryption key is known.

4.3) Summary of problem area

Android is fairly a new platform but growing very fast because of its intuitive and open nature of the architecture. The handset manufacturers have got the freedom to modify and this made them follow a successful business model. Along with this there comes the serious issue related to the criminal activities originating from these devices. Android forensic is still in its early stages of development and studies show the tools that claims to support the extraction of data from the Android devices are still limited in capability. However the extraction is still can be possible at the logical level. As like any phones Android devices also faces the small form factor limitation when comes to the forensic analysis. Examiners faces huge task to extract the data without any alteration to the target device. Forensic professionals tend to spend more time to understand how the tools work on different models of Android devices. They also need to verify the test data to ensure the result is expected or erroneous. There are both commercial and open source forensic tools available in the market but both of these tools not provide the 'forensically sound' extracted data. Many other methods of extraction should be tested and compared to pre-packaged software so the best option is selected.

5) Implementation of the technical track

5.1) Preparation of the Smartphone device for data acquisition

5.1.1) Rooting the Smartphone device: Samsung Galaxy S GT19000.

Rooting the Samsung Galaxy S GT19000.

First of all we have at our disposal the Smartphone device Samsung Galaxy S GT 19000. The device (by constructor) uses software version Android 2.3. We made software upgrade from Android 2.3 and placed the Cyanogenmod 10.2 (which is equivalent to the version Android 4.3). We had such an upgrade as only this would allow us to compile our own kernel and LiME could work (in 3rd scenario).

Rooting the Samsung Galaxy S GT19000 is quite easy to root it. Though there are many advantages of rooting, the only disadvantage being till now is that the warranty is voided as soon as we root the device, though the warranty can be claimed back once we un-root the device. After that we can get the administrator access after which right from changing the file system to removing the factory loaded apps can be enjoyed to name the few benefits.

There are two ways with which we can root your Samsung Galaxy S GT19000, one is by using the **Z4 Root** and the other one is by using the **Super One click root**. Both of these ways are very effective in rooting but some of the users have faced and reported some issues about rooting with the help of Z4 Root about device not being rooted successfully while for all those who wanted to simply root their phones temporarily or even permanently Z4 Root is only a good option.

First of all we make sure that we have all your applications backup, messages and also all our songs, images, videos and contacts as in the process of rooting we may lose all of this information, though it's an unlikely event that we lose everything. So, it's highly recommended to either sync all our data or create the backup of all our data so that we can restore the same after rooting or after the firmware update. So, let's proceed and see on how to root the device by two simple ways that is by using the Z4 Root as well as by using the Super One click Root.

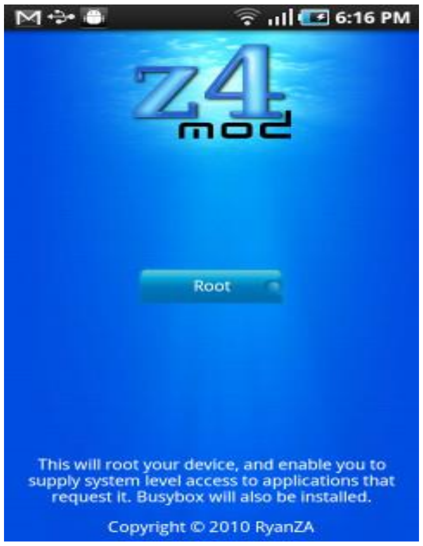
It's very important to maintain a minimum battery power of 50% so as to facilitate the smoother rooting process or else there is a high possibility that our device may end up turning completely off in the process of rooting. Since the process of rooting will be taking place, all the APN settings with which we access the Internet will be wiped off completely, so ensure that we have noted all the required settings for accessing the internet so that we can configure them again back once the device gets rooted.

Figure 65 Samsung Galaxy S GT19000

Procedure to Root Samsung Galaxy S GT 19000 by using Z4 Root:

- First of all we have to download the application called Z4 root and then we will have to unzip the same to get the .apk file which can be copied to our android phone easily. While unzipping the file if the password is prompted then you need to enter the password as "**androidadvices.com**". After the extraction we will get the apk file which is needed to be copied to Sony Xperia X10 Mini too.

- Now, as soon as we managed to copy the Z4Root's apk file in the phone, we need to go to the application list and just need to tap on the same to install it on the Samsung Galaxy S GT 19000.



This will root your device, and enable you to supply system level access to applications that request it. Busybox will also be installed.

Copyright © 2010 RyanZA

Figure 66 Z4 Root's apk file.

- Now, as soon as the installation of Z4 Root is completed, we will have to tap on the application to open the same after which we will see two options that is temporary root and another one will be the Permanent Root. From this, we will have to tap on the “Permanent Root” after which the device will be rooted in seconds. Note that while the process is running we should not interrupt its processes or else we will have to repeat the rooting process, and also its not recommended to choose the temporary root as if we do the temporary root then the device will get un-rooted just by a simple restart.

Figure 67 Z4 Root process.

- After selecting the appropriate option among the two, the rooting process will be finished and then the device will restart. If we are choosing the temporary root then the device will only stay rooted till the device is restarted as soon as we restart our phone the device will be un-rooted. So, it’s always recommended to select the Permanent Root. So, after the phone is restarted, a new icon of “**Superuser**” will be created in the screen which will be in the shape of skull. This means our device has been successfully rooted and finally we are ready to customize the device in the way we desire it.

Procedure to Root Samsung Galaxy S GT 19000 with Super One Click Root:

- In case that we haven’t succeeded in rooting our phone with the help of Z4 Root then by using **the Super One click Root**, we can easily root the Samsung Galaxy S GT 19000. This method too is just like the one click but this is software for personal computers and not for telephone’s software. There was another one click root solution which was popularly known as Z4 Root but due to some issues Google’s Android Market place has removed this application from the Android Market place. With its unique simple interface, we will never have to go through serious rooting capabilities like HRoot and etc. which sometimes is quite difficult to perform at times as a normal user.

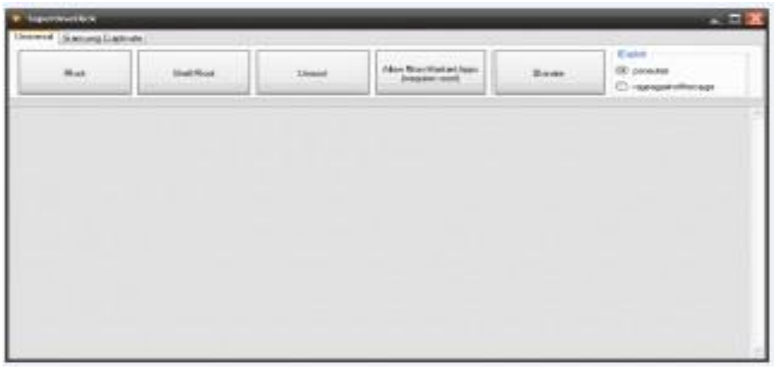


Figure 68 Super One Click Root.

- Before proceeding first, we will have to install the Java SDK, not just Runtime but the full SDK. The main reason to install this SDK is that to root the device from the computer will be requiring the drivers so that we can easily root the phone using the Super One click.

- To use this software application, first we will have to download the PC version of this software and then we will have to install the same. After installing, we need first to set our device in the USB debugging mode, making sure make sure that we do not mount the SD Card or else whole data will get lost or even some major issue can arise.

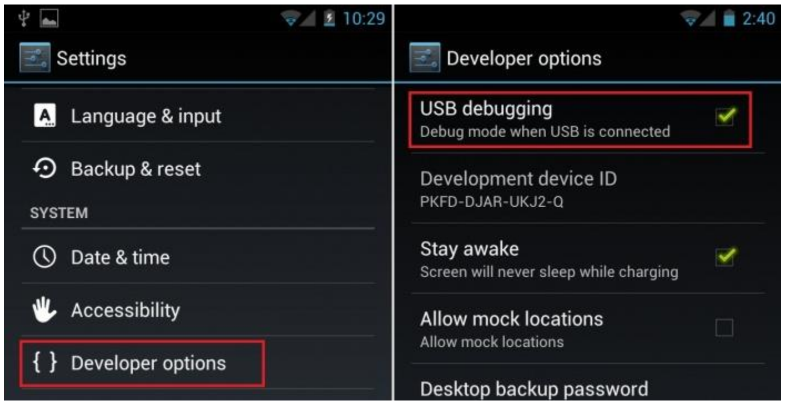
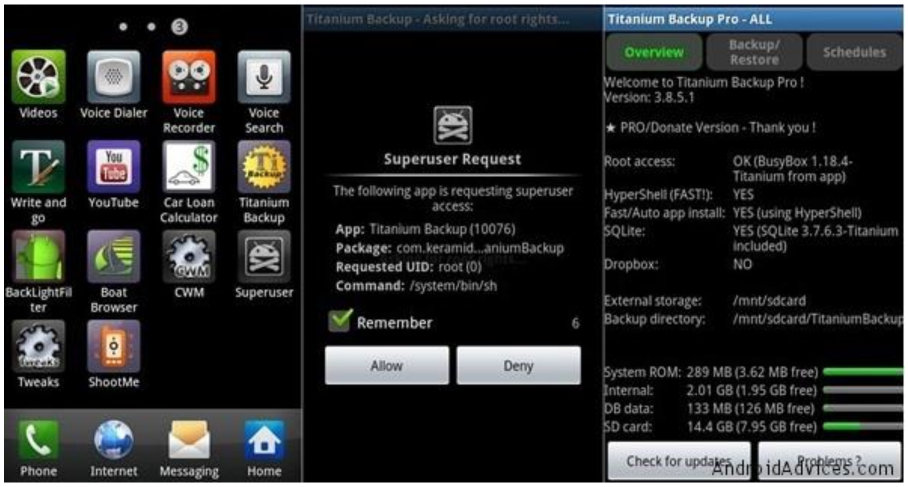


Figure 69 USB Debugging in Android Smartphone's screen.

- After this, we must open the installed Super One click software, connect the USB cable to our device (phone) and computer and if we are unable to get it run then we can try in the recovery mode. Make sure that all the drivers are installed through the PC suites of our respective Android phones. After connecting, all we need to do is just to select the option which we wish to perform and we have now successfully finished rooting, shell rooting, un-rooting as well as allowing us to install even Non-Market applications (requires rooting) very easily all in just one click.



```
Titanium Backup - Asking for root rights...

Superuser Request
The following app is requesting superuser access:
App: Titanium Backup (10076)
Package: com.keramid...aniumBackup
Requested UID: root (0)
Command: /system/bin/sh
```

```
Titanium Backup Pro - ALL
Welcome to Titanium Backup Pro !
Version: 3.8.5.1

★ PRO/Donate Version - Thank you !

Root access:            OK (BusyBox 1.18.4-
                        Titanium from app)
HyperShell (FAST!):     YES
Fast/Auto app install:  YES (using HyperShell)
SQLite:                 YES (SQLite 3.7.6.3-Titanium
                        included)
Dropbox:                NO

External storage:       /mnt/sdcard
Backup directory:       /mnt/sdcard/TitaniumBackup

System ROM: 289 MB (3.62 MB free)
Internal:   2.01 GB (1.95 GB free)
DB data:    133 MB (126 MB free)
SD card:    14.4 GB (7.95 GB free)
```

Figure 70 Super User on screen.

5.2) Scenarios of data acquisition

5.2.1) Applications

5.2.1.1) 1st Scenario: Root Browser

One possible way to recover the user's credentials, it is considered the case of the free tool type of Manager application (e.g. Root browser). For this reason, it was considered necessary to be tested the Root browser application. This application was downloaded from Android play store.

Root Browser is the Ultimate File Manager for rooted users. Explore all of Android's file systems and take control of our Android device.

The tool includes some features like:

- Two file manager panels,
- Batch copy/paste, zip, tar, delete, move any file or folder,
- Explore apk, rar, zip & jar files,
- Change file permissions and ownership,
- View and edit any file,
- sqlite explorer,
- Move, copy, rename, and delete files,
- Create and delete directories (folders),

- Send files by email,
- Add new files & folders in any directory,
- Install zips using clockwork recovery,
- Execute script files,
- Show list of files with thumbnails for images,
- Bookmark any folder,
- Open files and folders with other apps,
- Change the theme (double tap home button),
- Sort by name, size & date,
- Extract single files from zip/apks/jars,
- Search for files or folders.

Root browser is shown like this in all android Smartphone devices:

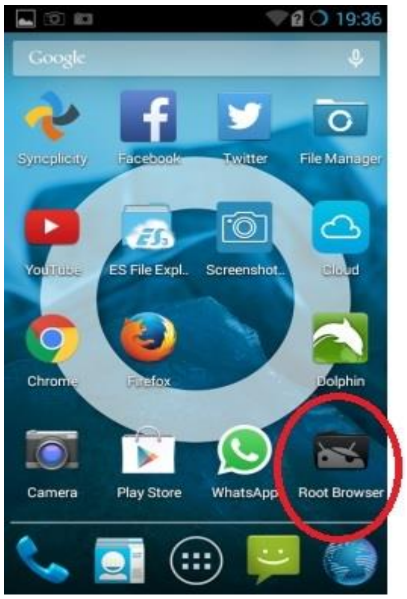
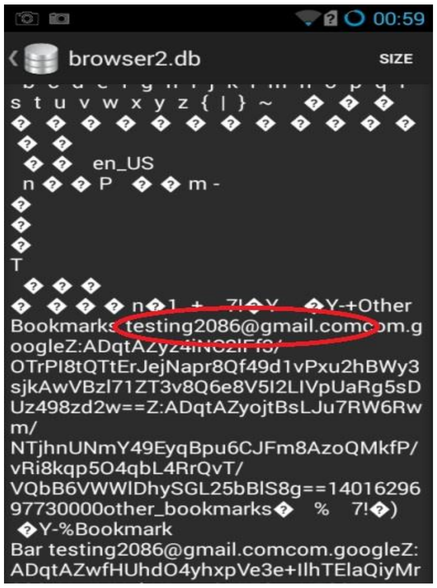


Figure 71 icon of Root Browser.

The application is able to represent files in the form of the whole system of the device with its contents. After a thorough navigation through the contents of the application, it was successfully recovered some of the credentials of the user (who had previously browsed) from the device's applications.

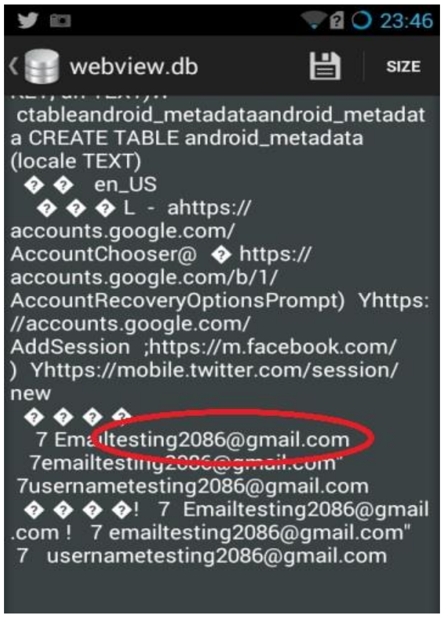
- First of all, the **default browser** of the android Smartphone had been tested. From the path: **/data/data/com.android.browser/databases** and especially the database "**browser2.db**" we found only the email account:



```
s t u v w x y z { | } ~   � � �
� � � � � � � � � � � �
 � �
  � �  en_US
 n � � P  � � m -
�
�
�
T
  � � �
�  � � � n�1 +  7!�Y   �Y-+Other
Bookmarks testing2086@gmail.comcom.g
oogleZ:ADqtAZy24iNC2IFf3/
OTrPI8tQTtErJejNapr8Qf49d1vPxu2hBWy3
sjkAwVBzl71ZT3v8Q6e8V5I2LIVpUaRg5sD
Uz498zd2w==Z:ADqtAZyojtBsLJu7RW6Rw
m/
NTjhnUNmY49EyqBpu6CJFm8AzoQMkfP/
vRi8kqp5O4qbL4RrQvT/
VQbB6VWWIDhySGL25bBIS8g==14016296
97730000other_bookmarks�   %   7!�)
 �Y-%Bookmark
Bar testing2086@gmail.comcom.googleZ:
ADqtAZwfHUhdO4yhxpVe3e+IlhTElaQiyMr
```

Figure 72

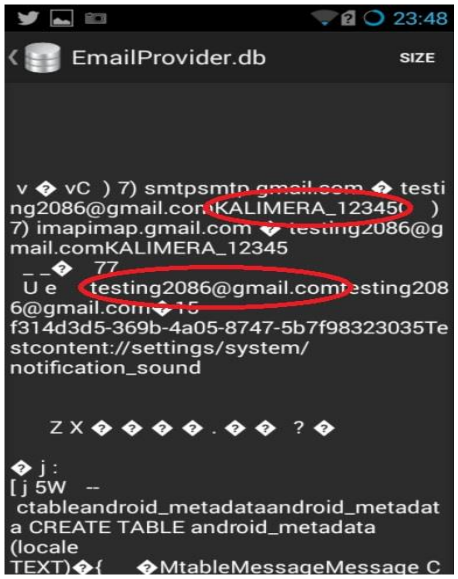
- And from the database "**webview.db**" (from the same path: **/data/data/com.android.browser/databases**), we have the same result as the same as the previous result:



```
 ctableandroid_metadataandroid_metadat
a CREATE TABLE android_metadata
(locale TEXT)
  � �  en_US
   � � � L -  ahttps://
accounts.google.com/
AccountChooser@  � https://
accounts.google.com/b/1/
AccountRecoveryOptionsPrompt)  Yhttps:
//accounts.google.com/
AddSession  ;https://m.facebook.com/
)  Yhttps://mobile.twitter.com/session/
new
  � � � �
   7 Emailtesting2086@gmail.com
  7emailtesting2086@gmail.com"
 7usernametesting2086@gmail.com
  � � � �!  7  Emailtesting2086@gmail
.com !  7 emailtesting2086@gmail.com"
 7   usernametesting2086@gmail.com
```

Figure 73

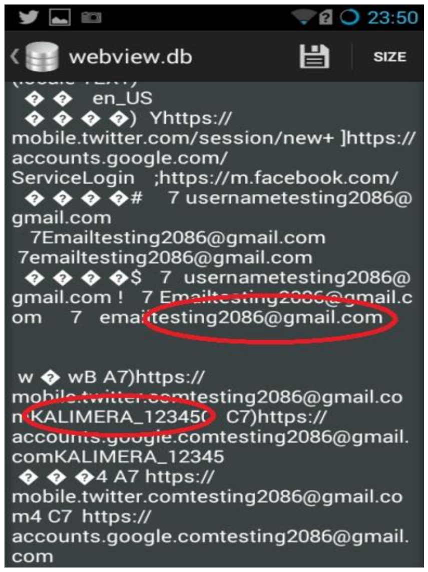
- Secondly, we tested the default **Android Email** and we retrieved these results from the database "**EmailProvider.db**" **(**which is located on the path: **data/data/com.android.email**):



```
v � vC ) 7) smtpsmtp.gmail.com � testi
ng2086@gmail.comKALIMERA_12345(   )
7) imapimap.gmail.com � testing2086@g
mail.comKALIMERA_12345
 _ _�   77
 U e   testing2086@gmail.comtesting208
6@gmail.com� 15
f314d3d5-369b-4a05-8747-5b7f98323035Te
stcontent://settings/system/
notification_sound


    Z X � � � � . � �  ? �

� j :
[ j 5W   --
 ctableandroid_metadataandroid_metadat
a CREATE TABLE android_metadata
(locale
TEXT)�{     �MtableMessageMessage C
```

Figure 74

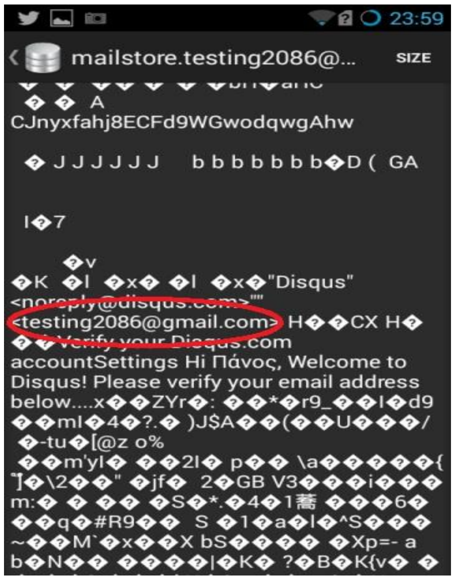
- Thirdly we checked a different internet browser **Cloud** from the path: **data/data/com.appdream.cloud/databases** and database "**webview.db**" we retrieved the credentials of the user (username and password):



```
  � �   en_US
  � � � �) Yhttps://
mobile.twitter.com/session/new+ ]https://
accounts.google.com/
ServiceLogin   ;https://m.facebook.com/
  � � � �#   7 usernametesting2086@
gmail.com
   7Emailtesting2086@gmail.com
  7emailtesting2086@gmail.com
  � � � �$   7  usernametesting2086@
gmail.com !   7 Emailtesting2086@gmail.c
om    7   emailtesting2086@gmail.com


 w � wB A7)https://
mobile.twitter.comtesting2086@gmail.co
mKALIMERA_12345(  C7)https://
accounts.google.comtesting2086@gmail.
comKALIMERA_12345
 � � �4 A7 https://
mobile.twitter.comtesting2086@gmail.co
m4 C7  https://
accounts.google.comtesting2086@gmail.
com
```

Figure 75

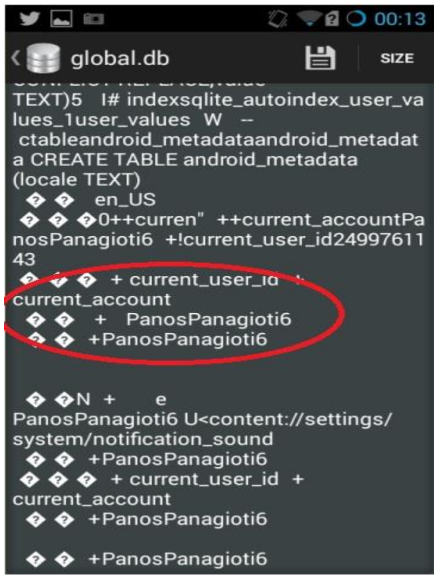
- We found only the email account from the email application **Gmail** from the path: **data/data/google.android.gm/databases** and the database: “**mailstore.testing2086@gmail.com.db**”:



```
 � �  A
CJnyxfahj8ECFd9WGwodqwgAhw

 � J J J J J J   b b b b b b b�D (  GA


 I�7

      �v
�K  �I  �x� �I  �x�"Disqus"
<noreply@disqus.com>""
<testing2086@gmail.com> H��CX H�
��Verify your Disqus.com
accountSettings Hi Πάνος, Welcome to
Disqus! Please verify your email address
below....x��ZYr�: ��*�r9_��I�d9
��mI�4�?.� )J$A��(��U����/
 �-tu�[@z o%
 ��m'yI� ��2I� p�� \a������{
‷]�\2��" �jf�  2�GB V3���i����
m:� � �� �S�*.�4�1蕎 ���6�
��q�#R9��  S �1�a�I�^S����
~��M`�x���X bS����� �Xp=- a
b�N�� �����|�K� ?�B�K{v� �
```

Figure 76

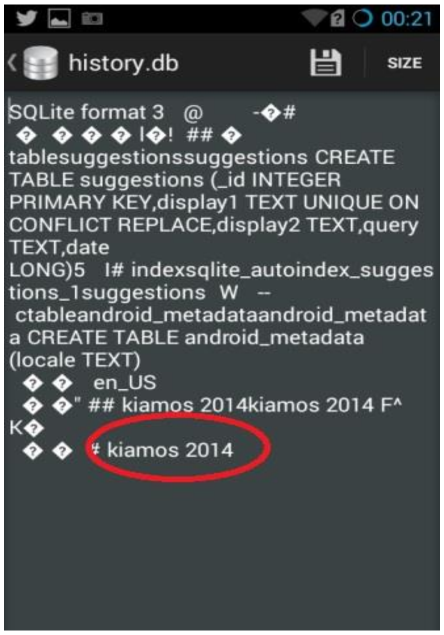
- From the Twitter application, the path: **data/data/com.twitter.android/databases** and the database: “**global.db**” we could find only the name of the user:



```
TEXT)5   I# indexsqlite_autoindex_user_va
lues_1user_values  W   --
 ctableandroid_metadataandroid_metadat
a CREATE TABLE android_metadata
(locale TEXT)
  � �   en_US
 � � �0++curren"  ++current_accountPa
nosPanagioti6  +!current_user_id24997611
43
 � � �  + current_user_id  +
current_account
  � �   +   PanosPanagioti6
  � �  +PanosPanagioti6


  � �N  +      e
PanosPanagioti6 U<content://settings/
system/notification_sound
  � �  +PanosPanagioti6
 � � �  + current_user_id  +
current_account
  � �  +PanosPanagioti6

  � �  +PanosPanagioti6
```

Figure 77

- Finally from the **Youtube** application, the database "**history.db**" and especially the path: **data/data/com.google.android.youtube/databases** we could find only a name of Greek singer:



```
SQLite format 3   @        -�#
 �  � � � l�!  ## �
tablesuggestionssuggestions CREATE
TABLE suggestions (_id INTEGER
PRIMARY KEY,display1 TEXT UNIQUE ON
CONFLICT REPLACE,display2 TEXT,query
TEXT,date
LONG)5   I# indexsqlite_autoindex_sugges
tions_1suggestions  W   --
 ctableandroid_metadataandroid_metadat
a CREATE TABLE android_metadata
(locale TEXT)
  � �   en_US
  � �" ## kiamos 2014kiamos 2014 F^
K�
  � �   # kiamos 2014
```

Figure 78

5.2.1.1.1) Summary

Many Mobile Forensic application (free or commercial) are able to reveal important information, data or credentials of user after using the device. So in this case the free tool Root Browser app) through checking (and experiments) recovered credentials (usernames and password) from applications on memory (ROM) of the mobile device. A Mobile Forensic application can be considered as a smart and alternative way to recover personal data (e.g. credentials) as the only precondition is to install it (application) in the device.

In conclusion, at the end of the experiment, it is worth to be noticed that the applications type of browser (e.g. Google Chrome, Mozilla Firefox and Cloud browser) revealed more information (e.g. usernames and passwords) except from the applications type of social media (e.g. Facebook, Twitter and Instagram), email (eg Gmail) and entertainment (eg Youtube).

5.2.2) Forensic tools

5.2.2.1) 2nd Scenario: Oxygen Forensic suite

Oxygen Forensic Suite 2014 belongs to the family of the commercial Mobile Forensic tools. The current software version provides access to the following sections: Phonebook, Calendar, Tasks, Messages, Event Log, File Browser and Extras (Life Blog, Phone Activity, Wi-Fi Connections, Skype and Web Cache analyzer). Note that the number of sections and list of extractable data fields depends on the device model.

Specially Oxygen Forensic Suite can extract a lot of unique information:

- Phone basic information and SIM-card data,
- Contacts list (including mobile, wireline, fax numbers, postal addresses, contact photos and other contact information),
- E-mail Messages with attachments,
- SMS Messages (messages, log, folders, deleted messages with some restrictions),
- Caller Groups information,
- All files from phone memory as well as from flash card, including installed applications and their data,
- Missed/Outgoing/Incoming calls,
- GPRS, EDGE, CSD, HSCSD and Wi-Fi traffic and sessions log,
- SIM card data,
- Text notes,
- Organizer (calendar meetings, appointments, memos, call reminders, anniversaries and birthdays, to-do tasks),
- Multimedia Messages with attachments,
- Photos and gallery images,
- Video clips and films,
- Voice records and audio clips,
- FM Radio Stations database (as a part of File Browser),
- Web browser cache and bookmarks,
- Lifeblog activity: all main events with geographical coordinates,
- Data Integrity protection with MD5, SHA-1, SHA-2, CRC, HAVAL, GOST Đ34.11-94.

Oxygen Forensic Suite 2014 offers an easy and convenient management of all examined devices in one window: phone properties, case details and status, the person in charge of it, etc.

Mobile device information analysis can be done from the program directly or with the help of advanced export function. We can create reports in the most popular file formats (XLS, RTF, PDF, XML, CSV, TSV) and either print or send them to remote departments and experts.

The program has a powerful built-in search engine. We can easily find the necessary information in all the sections with few mouse clicks in Oxygen Forensic Suite 2014. What is important, the search results are saved between sessions and can be either exported or printed. Besides, a contextual filter in every section helps you to sort out the data the way you need it.

Oxygen Forensic Suite 2014 contains two viewers that can open database as well as .plist files.

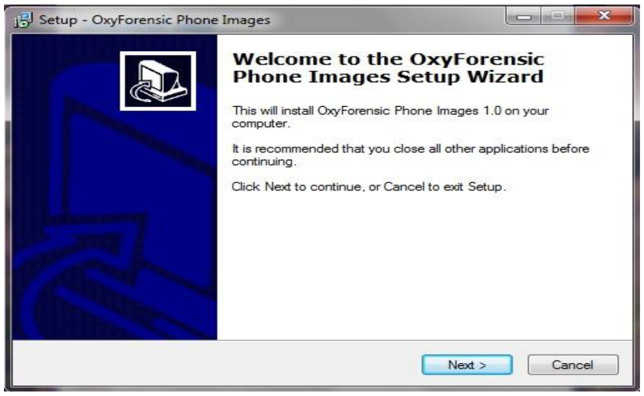
Moreover, the software allows you to save extracted data to a file and then load it into the program on another computer. Thus you need to connect a phone and extract data only once and then send the extracted information outside, e.g. for analysis by remote experts.

Current version works with more than 1600 mobile devices from Nokia, Apple, RIM (Blackberry), Google (based on Android OS), Samsung, Sony Ericsson, Motorola, Panasonic, LG, HTC, Asus, HP and other manufacturers. Oxygen Forensic Suite 2014 has a strong support for Symbian OS and Windows Mobile Smartphones (ActiveSync is not required).

The list of supported models is rapidly growing. Oxygen Forensic Suite 2014 supports USB cable connection, Bluetooth (Microsoft, Widcomm, BlueSoleil) connection, infrared connection using IrDA stack. Support for different types of connection depends on the phone series and model.

5.2.2.1.1) Installation

- First of all, we must run the **OxyForensic_Setup.exe** in order to start the installation of the tool.



Welcome to the OxyForensic Phone Images Setup Wizard

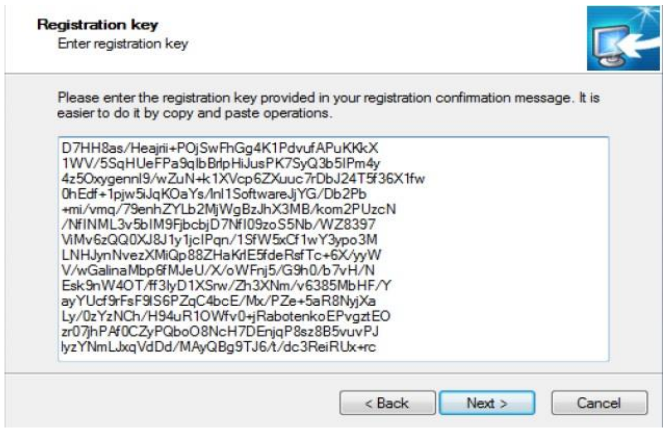
This will install OxyForensic Phone Images 1.0 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure 79

- Of course it is required an activation "key" for installation to be completed successfully:

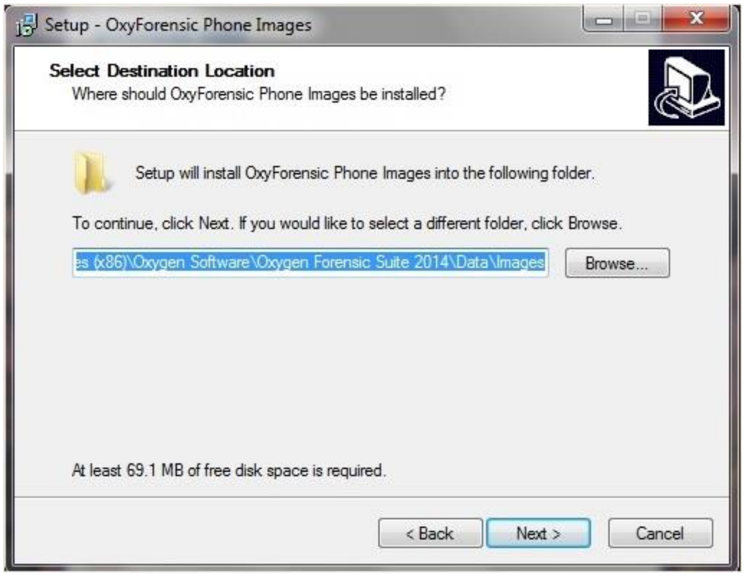

Registration key
Enter registration key

Please enter the registration key provided in your registration confirmation message. It is easier to do it by copy and paste operations.

```
D7HH8as/Heajrii+POjSwFhGg4K1PdvufAPuKKkX
1WV/5SqHUeFPa9qlbBrlpHiJusPK7SyQ3b5IPm4y
4z5Oxygennl9/wZuN+k1XVcp6ZXuuc7rDbJ24T5f36X1fw
0hEdf+1pjw5iJqKOaYs/lnl1SoftwareJjYG/Db2Pb
+mi/vmq/79enhZYLb2MjWgBzJhX3MB/kom2PUzcN
/NfINML3v5bIM9FjbcbjD7NfI09zoS5Nb/WZ8397
ViMv6zQQ0XJ8J1y1jcIPqn/1SfW5xCf1wY3ypo3M
LNHJynNvezXMiQp88ZHaKrlE5fdeRsfTc+6X/yyW
V/wGalinaMbp6fMJeU/X/oWFnj5/G9h0/b7vH/N
Esk9nW4OT/ff3lyD1XSrw/Zh3XNm/v6385MbHF/Y
ayYUcf9rFsF9lS6PZqC4bcE/Mx/PZe+5aR8NyjXa
Ly/0zYzNCh/H94uR1OWfv0+jRabotenkoEPvgztEO
zr07jhPAf0CZyPQboO8NcH7DEnjqP8sz8B5vuvPJ
lyzYNmLJxqVdDd/MAyQBg9TJ6/t/dc3ReiRUx+rc
```

Figure 80

- And finally it is asked in which path "phone images" can be stored:


Setup - OxyForensic Phone Images

Select Destination Location
Where should OxyForensic Phone Images be installed?

Setup will install OxyForensic Phone Images into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

es (x86)\Oxygen Software\Oxygen Forensic Suite 2014\Data\Images

At least 69.1 MB of free disk space is required.

Figure 81

- For starting the tool, we should do double-click on the following icon:

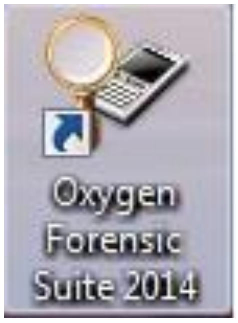
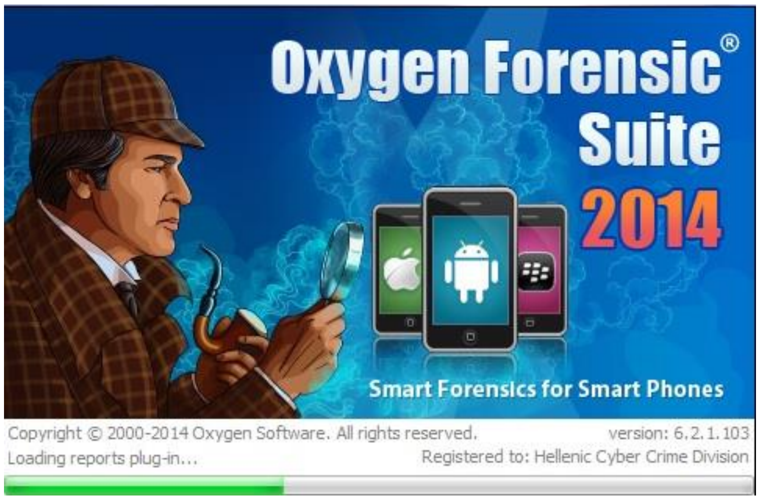


Figure 82

- Then the tool starts 'loading', according to the picture below:



Copyright © 2000-2014 Oxygen Software. All rights reserved. version: 6.2.1.103
Loading reports plug-in... Registered to: Hellenic Cyber Crime Division

Figure 83

- When the forensic tool finishes 'loading' then it starts working as shown in the picture below:

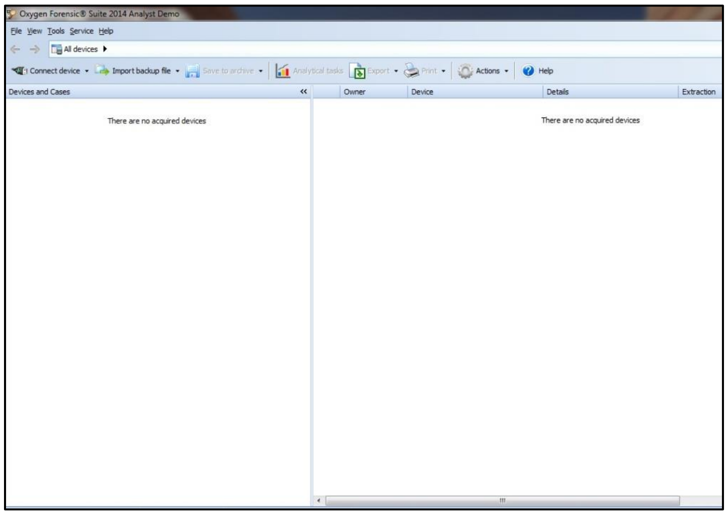


Figure 84

5.2.2.1.2) Hash functions

Data integrity with hash functions

Hash message authentication codes (HMAC) sign packets to verify that the information received is exactly the same as the information sent. This is called integrity. HMACs provide integrity through a keyed hash, the result of a mathematical calculation on a message using a hash function (algorithm) combined with a shared, secret key. A hash is commonly described as a signature on the packet. However, a hash differs from a digital signature. A hash uses a secret, shared key, and a digital signature uses public key technology and the private key of the sending computer. A digital signature provides nonrepudiation, and a hash does not. Nonrepudiation ensures that a communication can be proven to have originated from a specific person whose identity can be verified. It also ensures that the communication actually occurred.

Hash functions are also called one-way functions because it is easy to determine the hash from the message but mathematically infeasible to determine the message from the hash. Conversely, in two-way functions, the original message can be determined from its converted form. Encryption and decryption schemes are examples of two-way functions.

The hash is a cryptographic checksum or message integrity code (MIC) that each party must compute to verify the message. For example, the sending computer uses a hash function and shared key to compute the checksum for the message, including it with the packet. The receiving computer must perform the same hash function on the received message and shared key and compare it to the original (included in the packet from the sender). If the message has changed in transit, the hash values are different and the packet is rejected.

So for integrity, Oxygen Forensic suite can use six (6) hash functions when setting policy for exporting data:

MD5 hash algorithm

Message Digest 5 (MD5) is based on RFC 1321. MD5 completes four passes over the data blocks, using a different numeric constant for each word in the message on each pass. The number of 32-bit constants used during the MD5 computation ultimately produces a 128-bit hash that is used for the integrity check.

Secure Hash Algorithm (SHA)

The **Secure Hash Algorithm** is a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS), including:

- **SHA-1**: A 160-bit hash function which resembles the earlier MD5 algorithm. This was designed by the National Security Agency (NSA) to be part of the Digital Signature Algorithm. Cryptographic weaknesses were discovered in SHA-1, and the standard was no longer approved for most cryptographic uses after 2010.

- **SHA-2**: A family of two similar hash functions, with different block sizes, known as SHA-256 and SHA-512. They differ in the word size; SHA-256 uses 32-bit words where SHA-512 uses 64-bit words. There are also truncated versions of each standard, known as SHA-224 and SHA-384. These were also designed by the NSA.

University of Piraeus

GOST R 34.11-94 hash algorithm

The **GOST hash function**, defined in the standards **GOST R 34.11-94** and **GOST 34.311-95**, is a 256-bitcryptographic hash function. It was initially defined in the Russian national standard GOST R 34.11-94 *Information Technology - Cryptographic Information Security - Hash Function*. The equivalent standard used by other member-states of the CIS is GOST 34.311-95.

The **GOST hash function**, defined in the standards **GOST R 34.11-94** and **GOST 34.311-95**, is a 256-bitcryptographic hash function. It was initially defined in the Russian national standard GOST R 34.11-94 *Information Technology - Cryptographic Information Security - Hash Function*. The equivalent standard used by other member-states of the CIS is GOST 34.311-95.

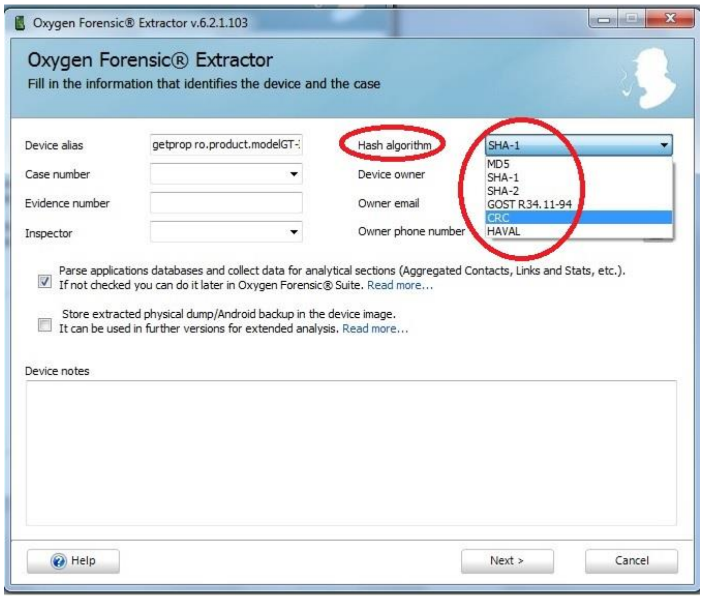
CRC has algorithm

A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents; on retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match.

CRCs are so called because the *check* (data verification) value is a *redundancy* (it expands the message without adding information) and the algorithm is based on *cyclic* codes. CRCs are popular because they are simple to implement in binary hardware, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function.

Haval hash algorithm

HAVAL is a cryptographic hash function. Unlike MD5, but like most modern cryptographic hash functions, HAVAL can produce hashes of different lengths. HAVAL can produce hashes in lengths of 128 bits, 160 bits, 192 bits, 224 bits, and 256 bits. HAVAL also allows users to specify the number of rounds (3, 4, or 5) to be used to generate the hash.


Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor

Fill in the information that identifies the device and the case

Device alias: getprop ro.product.modelGT-

Case number

Evidence number

Inspector

Hash algorithm: SHA-1

MD5
SHA-1
SHA-2
GOST R34.11-94
CRC
HAVAL

Device owner

Owner email

Owner phone number

Parse applications databases and collect data for analytical sections (Aggregated Contacts, Links and Stats, etc.).
If not checked you can do it later in Oxygen Forensic® Suite. Read more...

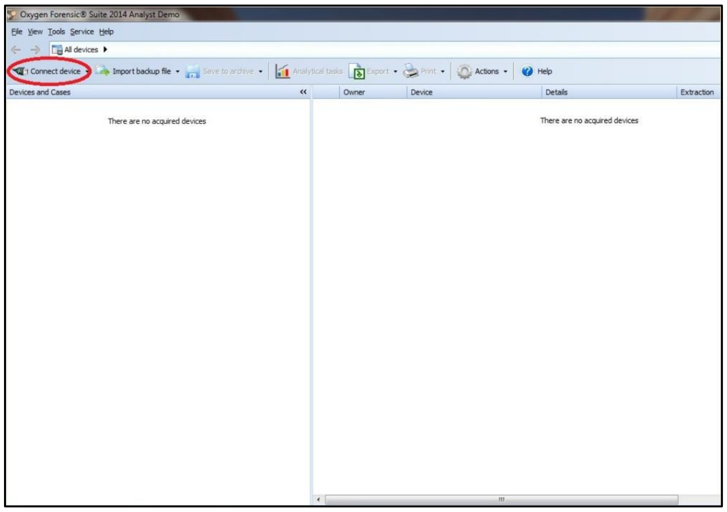
Store extracted physical dump/Android backup in the device image.
It can be used in further versions for extended analysis. Read more...

Device notes

Figure 85

5.2.2.1.3) Connection device

- In order to connect the device with Oxygen tool, we should select the button: "**Connect Device**" from the home screen as shown in the following picture:

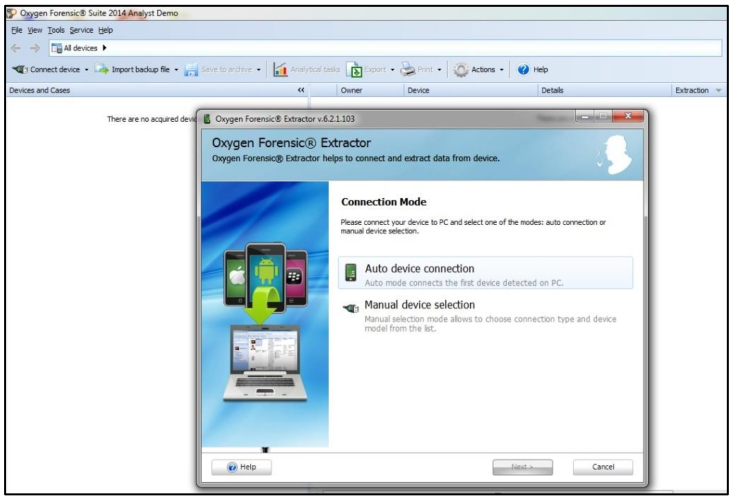

Oxygen Forensic® Suite 2014 Analyst Demo

There are no acquired devices

There are no acquired devices

Figure 86

- We are asked to choose which type of connection we desire:


Oxygen Forensic® Suite 2014 Analyst Demo

File View Tools Service Help

All devices

Connect device · Import backup file · Save to archive · Analytical tasks · Export · Print · Actions · Help

Devices and Cases | Owner | Device | Details | Extraction

There are no acquired devic

Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor

Oxygen Forensic® Extractor helps to connect and extract data from device.

Connection Mode

Please connect your device to PC and select one of the modes: auto connection or manual device selection.

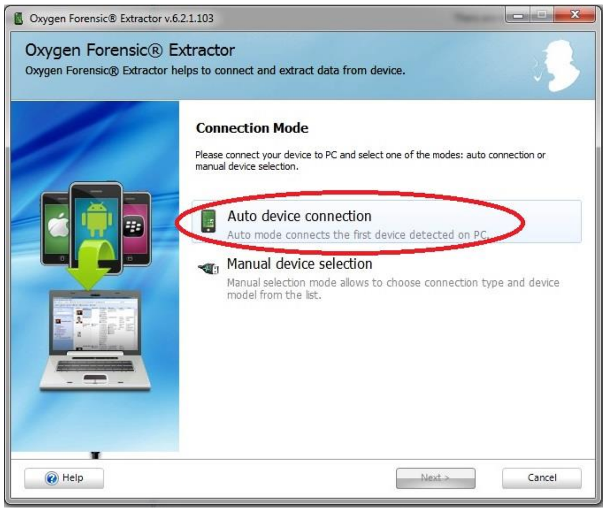
Auto device connection
Auto mode connects the first device detected on PC.

Manual device selection
Manual selection mode allows to choose connection type and device model from the list.

Help · Next > · Cancel

Figure 87

- We pick the choice "**Auto device connection**":


Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor

Oxygen Forensic® Extractor helps to connect and extract data from device.

Connection Mode

Please connect your device to PC and select one of the modes: auto connection or manual device selection.

Auto device connection
Auto mode connects the first device detected on PC.

Manual device selection
Manual selection mode allows to choose connection type and device model from the list.

Help · Next > · Cancel

Figure 88

- The program will look for recognizing the type of mobile device:

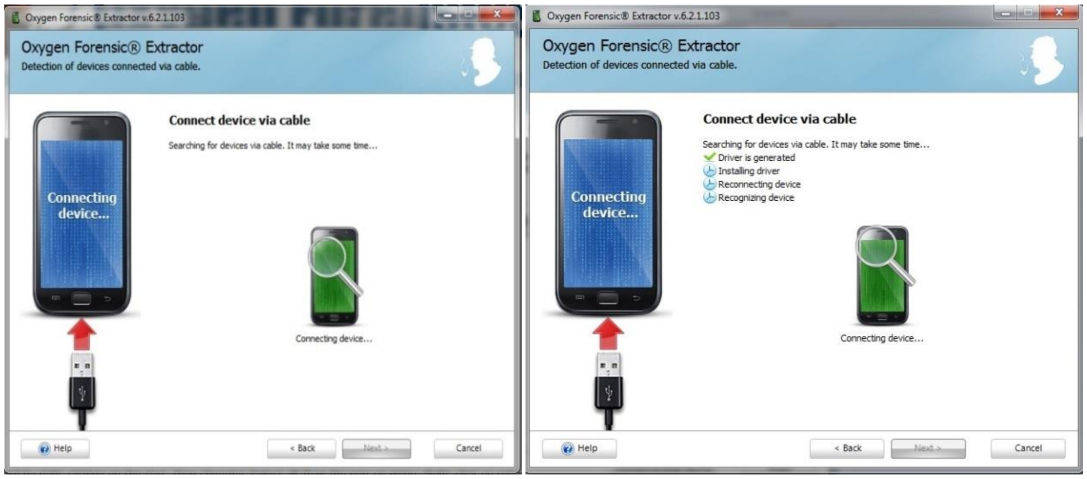

Oxygen Forensic® Extractor v.6.2.1.103
Oxygen Forensic® Extractor
Detection of devices connected via cable.
Connect device via cable
Searching for devices via cable. It may take some time...
Driver is generated
Installing driver
Reconnecting device
Recognizing device
Connecting device...
Help
< Back
Next >
Cancel

Figure 89

- Before, it is advisable to have allowed the choice "USB debugging" in the device. This choice does not allow having warnings like the picture below:

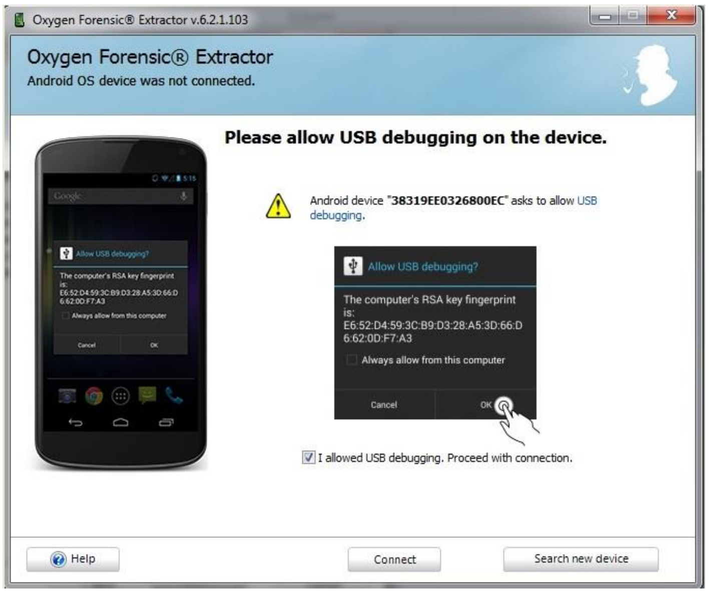
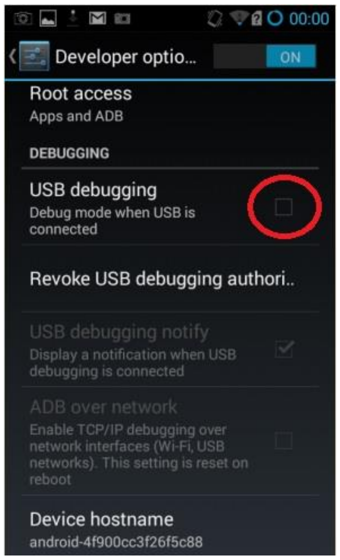

Oxygen Forensic® Extractor v.6.2.1.103
Oxygen Forensic® Extractor
Android OS device was not connected.
Please allow USB debugging on the device.
Android device "38319EE0326800EC" asks to allow USB debugging.
Allow USB debugging?
The computer's RSA key fingerprint is:
E6:52:D4:59:3C:B9:D3:28:A5:3D:66:D6:62:0D:F7:A3
Always allow from this computer
Cancel
OK
I allowed USB debugging. Proceed with connection.
Help
Connect
Search new device

Figure 90

- So from the path **System settings→ Developer options→ USB debugging** is activated:



00:00

Developer optio... ON

Root access
Apps and ADB

DEBUGGING

USB debugging
Debug mode when USB is connected

Revoke USB debugging authori..

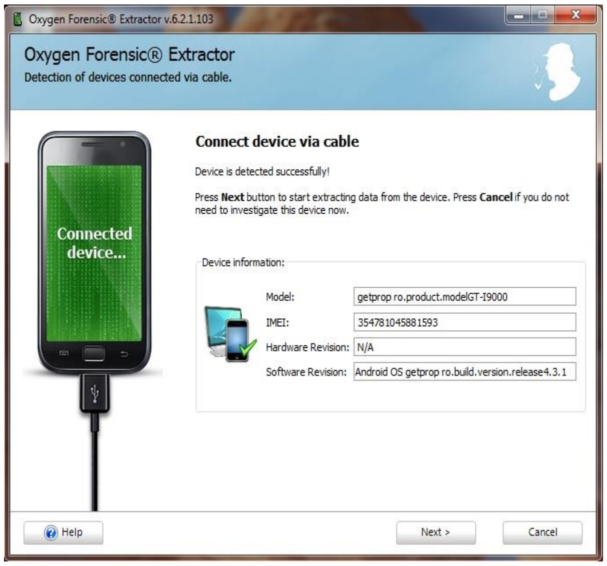
USB debugging notify
Display a notification when USB debugging is connected

ADB over network
Enable TCP/IP debugging over network interfaces (Wi-Fi, USB networks). This setting is reset on reboot

Device hostname
android-4f900cc3f26f5c88

Figure 91

- And there are two possible cases, either to recognize the device (see the left icon) or not detect the device and display an error message (see the right one):



Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor
Detection of devices connected via cable.

Connect device via cable

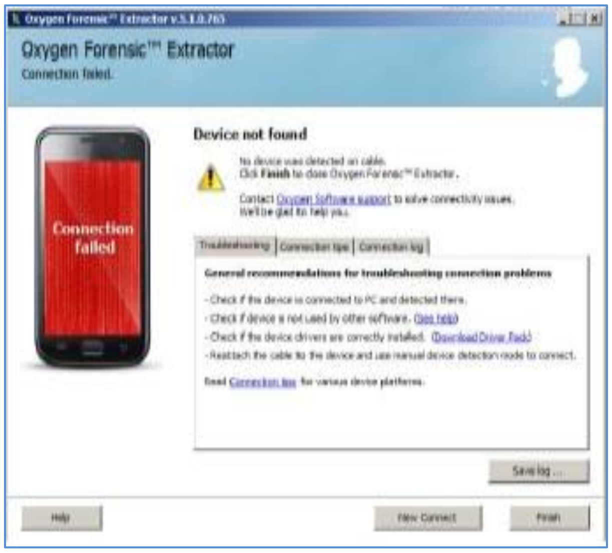
Device is detected successfully!

Press **Next** button to start extracting data from the device. Press **Cancel** if you do not need to investigate this device now.

Device information:

Model:	getprop ro.product.modelGT-I9000
IMEI:	354781045881593
Hardware Revision:	N/A
Software Revision:	Android OS getprop ro.build.version.release4.3.1

Figure 92



Oxygen Forensic™ Extractor v.5.1.0.765

Oxygen Forensic™ Extractor
Connection failed.

Device not found

No device was detected on cable.
Click **Finish** to close Oxygen Forensic™ Extractor.

Contact Oxygen Software support to solve connectivity issues.
We'll be glad to help you.

General recommendations for troubleshooting connection problems

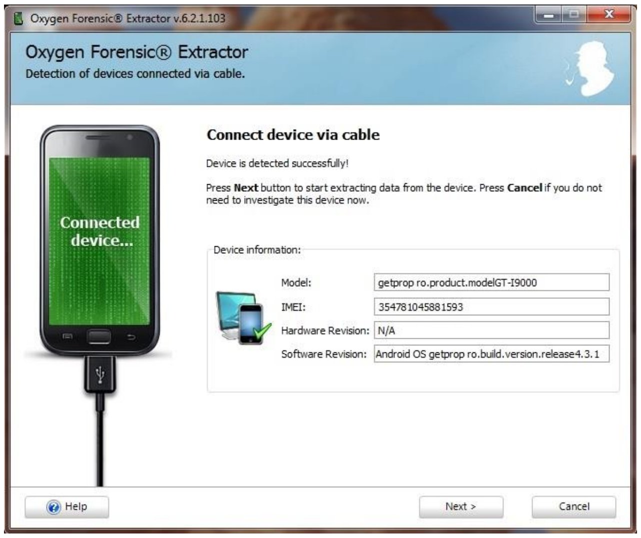
- Check if the device is connected to PC and detected there.
- Check if device is not used by other software. (See help)
- Check if the device drivers are correctly installed. (Download Driver Pack)
- Reattach the cable to the device and use manual device detection mode to connect.

Read Connection tips for various device platforms.

Figure 93

5.2.2.1.4) Export Data

- Once the device will be recognized, then tool provides information about the connected device including model, IMEI and version of the device and its software:



Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor
Detection of devices connected via cable.

Connect device via cable

Device is detected successfully!

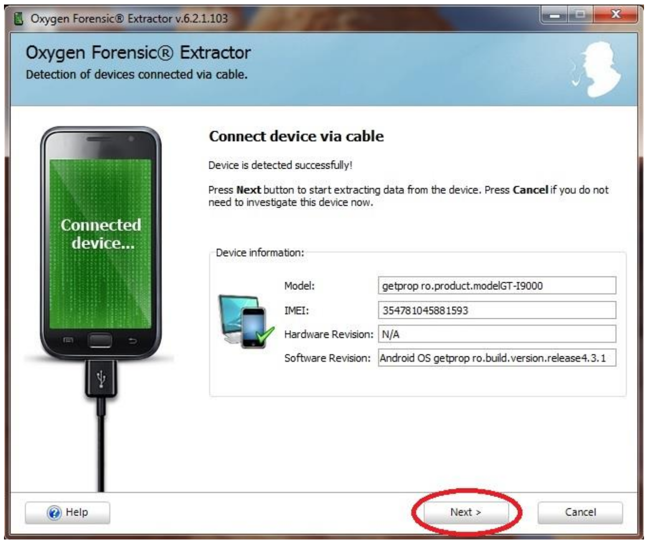
Press **Next** button to start extracting data from the device. Press **Cancel** if you do not need to investigate this device now.

Device information:

Model:	getprop ro.product.modelGT-I9000
IMEI:	354781045881593
Hardware Revision:	N/A
Software Revision:	Android OS getprop ro.build.version.release4.3.1

Figure 94

- We click on the button “Next” so as to move to the process of exporting data:



Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor
Detection of devices connected via cable.

Connected device...

Connect device via cable

Device is detected successfully!

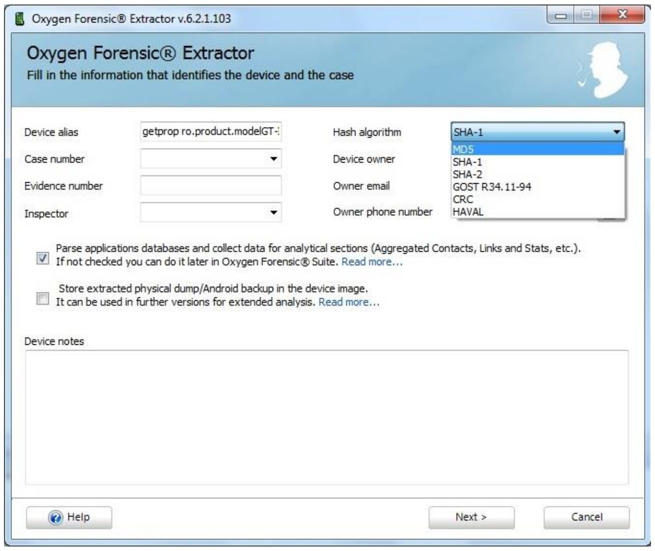
Press **Next** button to start extracting data from the device. Press **Cancel** if you do not need to investigate this device now.

Device information:

Model:	getprop ro.product.modelGT-I9000
IMEI:	354781045881593
Hardware Revision:	N/A
Software Revision:	Android OS getprop ro.build.version.release4.3.1

Figure 95

- The picture below, provide us a form in order to be filled with the necessary on the case (device alias, case number, evidence number, inspector, hash algorithm, device owner, owner email and owner phone number):



Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor
Fill in the information that identifies the device and the case

Device alias	getprop ro.product.modelGT-I	Hash algorithm	SHA-1
Case number		Device owner	
Evidence number		Owner email	
Inspector		Owner phone number	

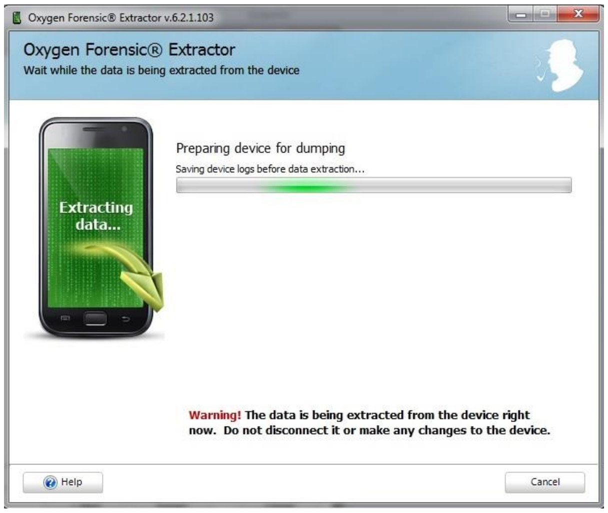
☑ Parse applications databases and collect data for analytical sections (Aggregated Contacts, Links and Stats, etc.).
If not checked you can do it later in Oxygen Forensic® Suite. Read more...

☐ Store extracted physical dump/Android backup in the device image.
It can be used in further versions for extended analysis. Read more...

Device notes

Figure 96

- The device is preparing by saving the logs and a new case will be created before the phase of data extraction:


Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor

Wait while the data is being extracted from the device

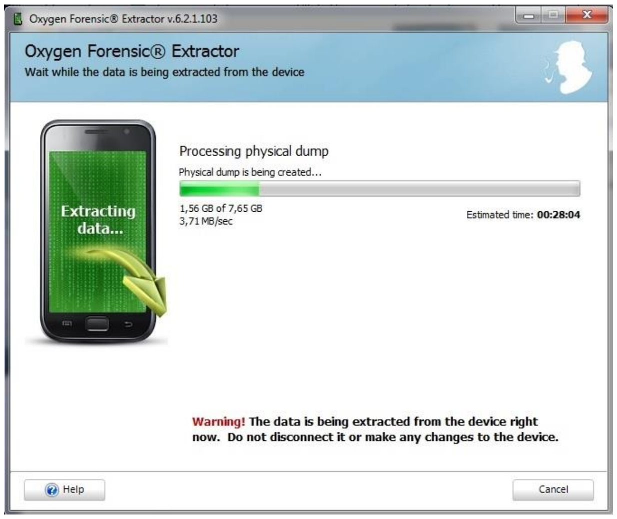
Preparing device for dumping

Saving device logs before data extraction...

Warning! The data is being extracted from the device right now. Do not disconnect it or make any changes to the device.

Figure 97

- The tool is in the phase of creating the physical dump memory:


Oxygen Forensic® Extractor v.6.2.1.103

Oxygen Forensic® Extractor

Wait while the data is being extracted from the device

Processing physical dump

Physical dump is being created...

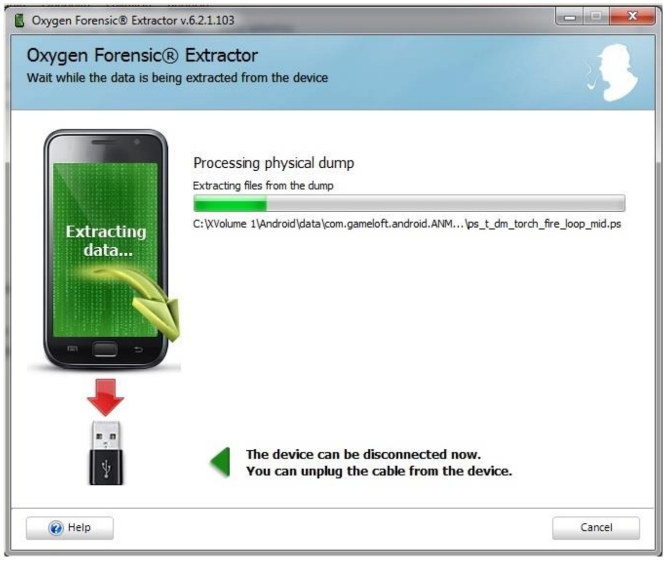
1,56 GB of 7,65 GB
3,71 MB/sec

Estimated time: **00:28:04**

Warning! The data is being extracted from the device right now. Do not disconnect it or make any changes to the device.

Figure 98

- Now it files are being extracted from the physical dump memory:



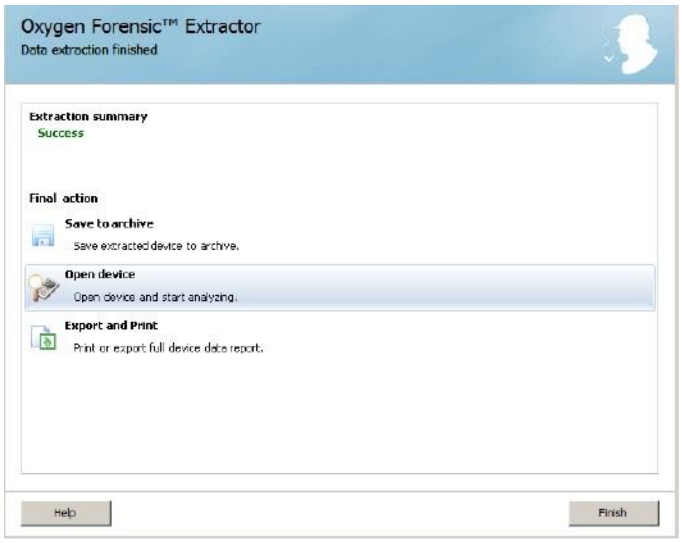
Oxygen Forensic® Extractor
Wait while the data is being extracted from the device

Processing physical dump
Extracting files from the dump
C:\XVolume 1\Android\data\com.gameloft.android.ANM...\ps_t_dm_torch_fire_loop_mid.ps

The device can be disconnected now.
You can unplug the cable from the device.

Figure 99

- Having completed successfully the export then options are given to save the file created or open device data and begin analysis or export a detailed report and print it:



Oxygen Forensic™ Extractor
Data extraction finished

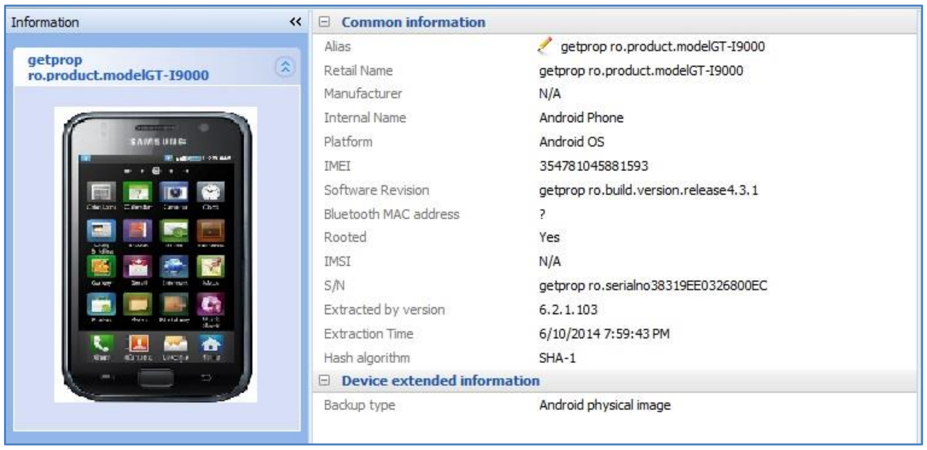
Extraction summary
Success

Final action
Save to archive
Save extracted device to archive.
Open device
Open device and start analyzing.
Export and Print
Print or export full device data report.

Figure 100

5.2.2.1.5) Oxygen forensic Acquisition

- Initially it is possible to see significant information of the under consideration mobile device:


Information

getprop ro.product.modelGT-I9000

Common information

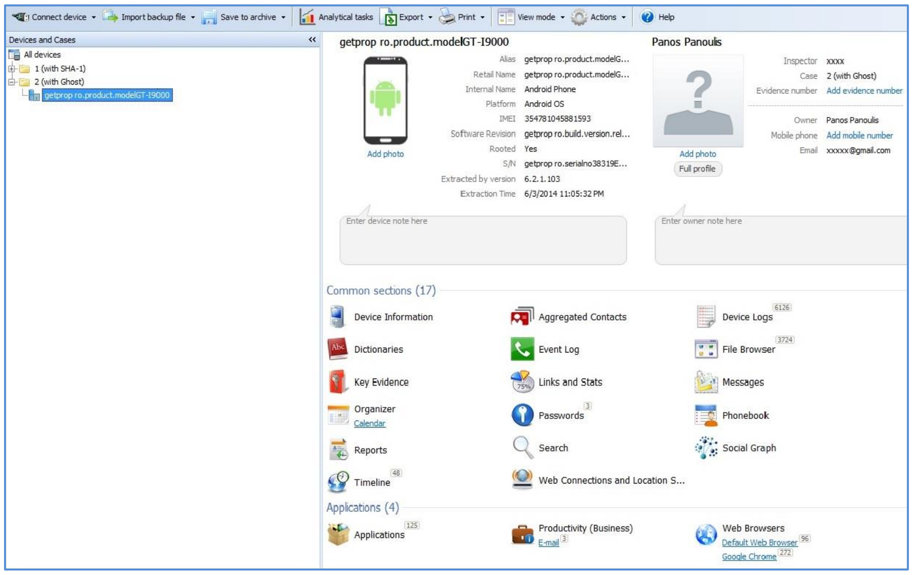
Alias	getprop ro.product.modelGT-I9000
Retail Name	getprop ro.product.modelGT-I9000
Manufacturer	N/A
Internal Name	Android Phone
Platform	Android OS
IMEI	354781045881593
Software Revision	getprop ro.build.version.release4.3.1
Bluetooth MAC address	?
Rooted	Yes
IMSI	N/A
S/N	getprop ro.serialno38319EE0326800EC
Extracted by version	6.2.1.103
Extraction Time	6/10/2014 7:59:43 PM
Hash algorithm	SHA-1

Device extended information

Backup type	Android physical image

Figure 101

- By choosing to navigate to the device exported data and start looking for, it is shown the below image. Now it's time the phase of examining data:



Devices and Cases

All devices
1 (with SHA-1)
2 (with Ghost)
getprop ro.product.modelGT-I9000

getprop ro.product.modelGT-I9000

Alias	getprop ro.product.modelG...
Retail Name	getprop ro.product.modelG...
Internal Name	Android Phone
Platform	Android OS
IMEI	354781045881593
Software Revision	getprop ro.build.version.rel...
Rooted	Yes
S/N	getprop ro.serialno38319E...
Extracted by version	6.2.1.103
Extraction Time	6/3/2014 11:05:32 PM

Panos Panoulis

Inspector	xxxx
Case	2 (with Ghost)
Evidence number	Add evidence number
Owner	Panos Panoulis
Mobile phone	Add mobile number
Email	xxxxx@gmail.com

Common sections (17)

Device Information; Aggregated Contacts; Device Logs 6126; Dictionaries; Event Log; File Browser 3724; Key Evidence; Links and Stats; Messages; Organizer (Calendar); Passwords 3; Phonebook; Reports; Search; Social Graph; Timeline 48; Web Connections and Location S...

Applications (4)

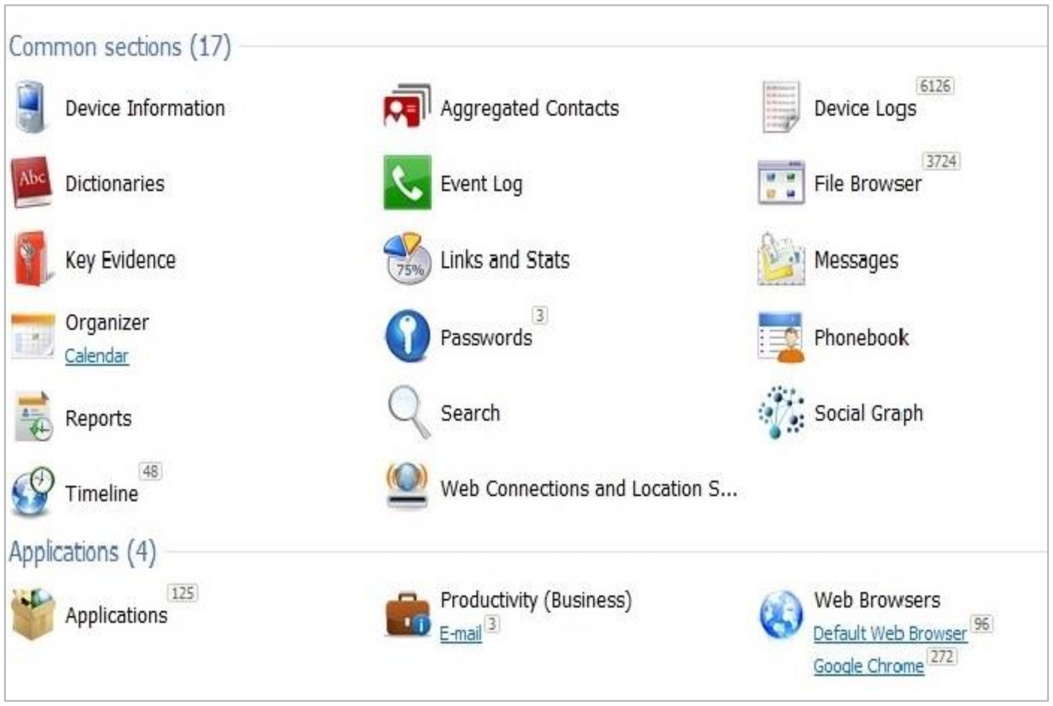
Applications 125; Productivity (Business) (E-mail 3); Web Browsers (Default Web Browser 96, Google Chrome 272)

Figure 102

The test was performed on mobile phone Samsung Galaxy S and Oxygen Forensics Suite 2014 was the tool which was selected. As for the export of data the method of logical extraction was followed.

It was seemed appropriate, our cell to be empty of personal messages, calls and photos. All the attention was focused on recovery credential data from the applications (e.g. the Social media and browsers) of device.

- Detailed data and information extracted illustrated in the figure below, which are analyzed in the following table:



Common sections (17)

Device Information
Aggregated Contacts
Device Logs 6126
Dictionaries
Event Log
File Browser 3724
Key Evidence
Links and Stats
Messages
Organizer
Calendar
Passwords 3
Phonebook
Reports
Search
Social Graph
Timeline 48
Web Connections and Location S...

Applications (4)

Applications 125
Productivity (Business)
E-mail 3
Web Browsers
Default Web Browser 96
Google Chrome 272

Figure 103

- The following table shows how many items the tool Oxygen Suite 2014 managed retrieved from our mobile device.

Applications	125	Device Logs	6126
Productivity (bussiness)	3	File Browser	3724
Web Browser	96	Timeline	48
Passwords	3		

In the next chapters, it will be analyzed per section and category findings and retrieved information from the forensic examination.

5.2.2.1.6) Data analysis of mobile device

5.2.2.1.6.1) Applications

Oxygen Forensic Suite retrieves numerous application data from a mobile device. In the Applications section, forensic experts view the list of pre-installed and user applications with the files created by these programs.

Each application can contain valuable user data, like passwords, logs, history, files and so on.

Section offers the following main features:

- Get logins and passwords to the app
- Find geo-location of the last run
- Inspect all used or created app files
- Know exactly when the app was used
- Access to system and user apps
- Filter apps by a certain term
- Export and print selected items

Many popular applications have a special User Data data tab where investigators find application data categorized and prepared for effective analysis.

Forensic experts can always access source files to learn how Oxygen Forensic Suite gathers information for User Data tab or to analyze applications that were not automatically prepared.

Oxygen Forensic Suite is the only smart phone forensics software that allows analyzing Applications in such a deep and structured way.

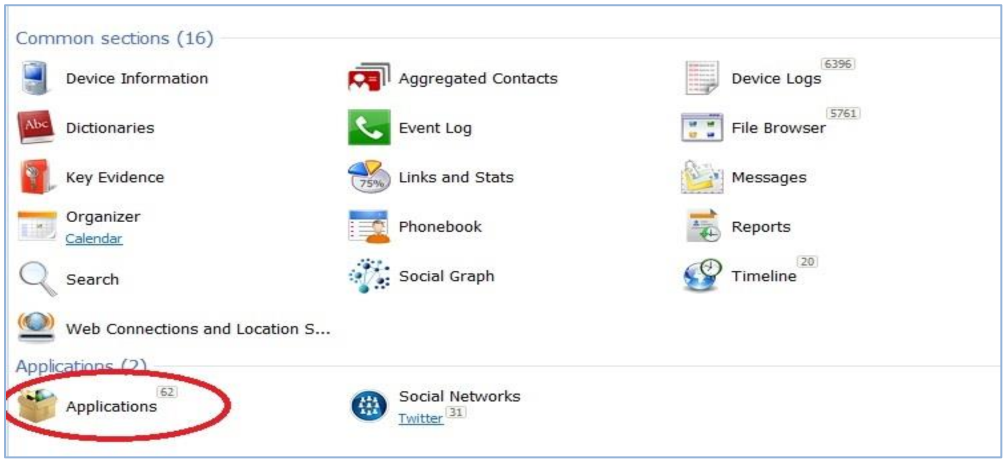
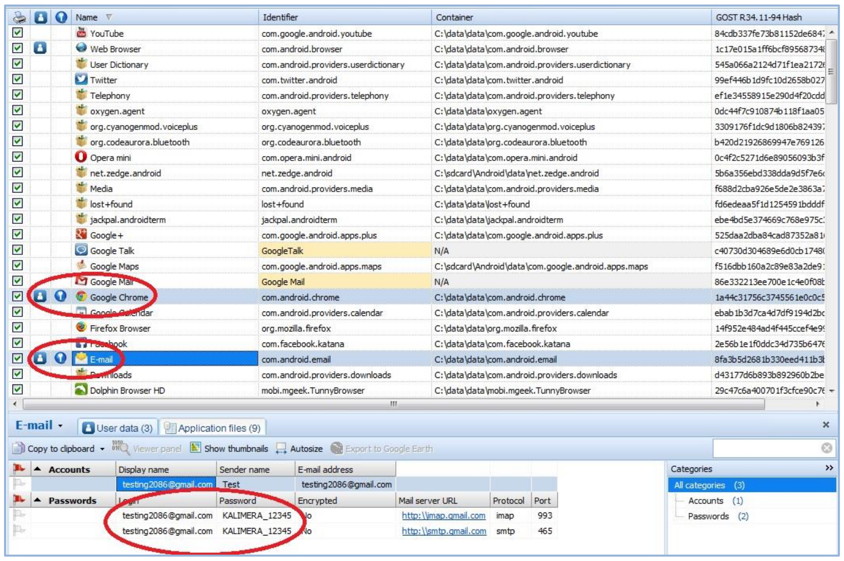


Figure 104

- As it seems in the following figure only two applications (Google Chrome and default Android email) recovered the credentials of the user:

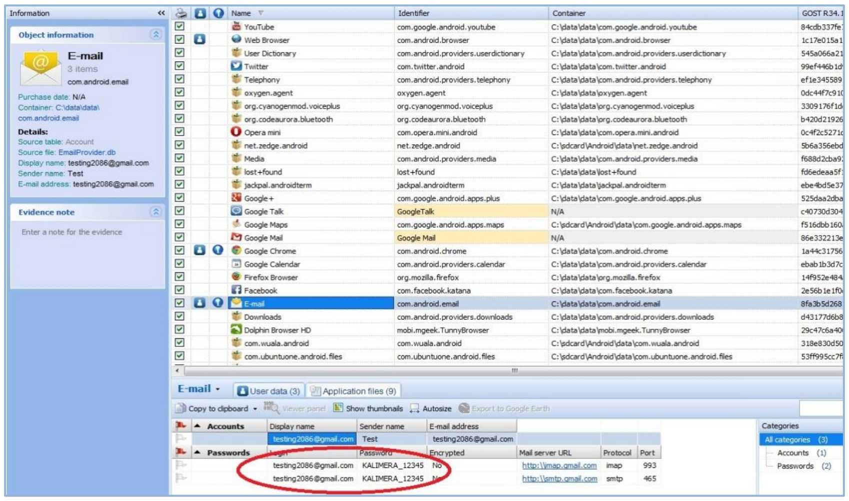


			Name	Identifier	Container	GOST R34.11-94 Hash
✓			YouTube	com.google.android.youtube	C:\data\data\com.google.android.youtube	84cdb337fe73b81152de684
✓	●		Web Browser	com.android.browser	C:\data\data\com.android.browser	1c17e015a1ff6bcf8956873
✓			User Dictionary	com.android.providers.userdictionary	C:\data\data\com.android.providers.userdictionary	545a066a2124d71f1ea2172
✓			Twitter	com.twitter.android	C:\data\data\com.twitter.android	99ef446b1d9fc10d2658b027
✓			Telephony	com.android.providers.telephony	C:\data\data\com.android.providers.telephony	ef1e34558915e290d4f20cdd
✓			oxygen.agent	oxygen.agent	C:\data\data\oxygen.agent	0dc44f7c910874b118f1aa05
✓			org.cyanogenmod.voiceplus	org.cyanogenmod.voiceplus	C:\data\data\org.cyanogenmod.voiceplus	3309176f1dc9d1806b82439
✓			org.codeaurora.bluetooth	org.codeaurora.bluetooth	C:\data\data\org.codeaurora.bluetooth	b420d21926869947e769126
✓			Opera mini	com.opera.mini.android	C:\data\data\com.opera.mini.android	0c4f2c5271d6e89056093b3f
✓			net.zedge.android	net.zedge.android	C:\sdcard\Android\data\net.zedge.android	5b6a356ebd338dda9d5f7e6
✓			Media	com.android.providers.media	C:\data\data\com.android.providers.media	f688d2cba926e5de2e3863a
✓			lost+found	lost+found	C:\data\data\lost+found	fd6edeaa5f1d1254591bdddf
✓			jackpal.androidterm	jackpal.androidterm	C:\data\data\jackpal.androidterm	ebe4bd5e374669c768e975c
✓			Google+	com.google.android.apps.plus	C:\data\data\com.google.android.apps.plus	525daa2dba84cad87352a81
✓			Google Talk	GoogleTalk	N/A	c40730d304689e6d0cb1748
✓			Google Maps	com.google.android.apps.maps	C:\sdcard\Android\data\com.google.android.apps.maps	f516dbb160a2c89e83a2de9
✓			Google Mail	Google Mail	N/A	86e332213ee700e1c4e0f08
✓	●	●	Google Chrome	com.android.chrome	C:\data\data\com.android.chrome	1a44c31756c3745561e0c0c
✓			Google Calendar	com.android.providers.calendar	C:\data\data\com.android.providers.calendar	ebab1b3d7ca4d7df9194d2b
✓			Firefox Browser	org.mozilla.firefox	C:\data\data\org.mozilla.firefox	14f952e484ad4f445ccef4e
✓			Facebook	com.facebook.katana	C:\data\data\com.facebook.katana	2e56b1e1f0ddc34d735b647
✓	●	●	E-mail	com.android.email	C:\data\data\com.android.email	8fa3b5d2681b330eed411b3
✓			Downloads	com.android.providers.downloads	C:\data\data\com.android.providers.downloads	d43177d6b893b892960b2be
✓			Dolphin Browser HD	mobi.mgeek.TunnyBrowser	C:\data\data\mobi.mgeek.TunnyBrowser	29c47c6a400701f3cfce90c7

Accounts	Display name	Sender name	E-mail address			
	testing2086@gmail.com	Test	testing2086@gmail.com			
Passwords	Login	Password	Encrypted	Mail server URL	Protocol	Port
	testing2086@gmail.com	KALIMERA_12345	No	http:\\imap.gmail.com	imap	993
	testing2086@gmail.com	KALIMERA_12345	No	http:\\smtp.gmail.com	smtp	465

Figure 105

- Specifically the default Android email application retrieved the credentials (username and password) of the user:



Information

Object information

E-mail
3 items
com.android.email

Purchase date: N/A
Container: C:\data\data\com.android.email

Details:
Source table: Account
Source file: EmailProvider.db
Display name: testing2086@gmail.com
Sender name: Test
E-mail address: testing2086@gmail.com

Evidence note

Enter a note for the evidence

Name	Identifier	Container	GOST R34.1
YouTube	com.google.android.youtube	C:\data\data\com.google.android.youtube	84cdb337fe
Web Browser	com.android.browser	C:\data\data\com.android.browser	1c17e015a1
User Dictionary	com.android.providers.userdictionary	C:\data\data\com.android.providers.userdictionary	545a066a21
Twitter	com.twitter.android	C:\data\data\com.twitter.android	99ef446b1d
Telephony	com.android.providers.telephony	C:\data\data\com.android.providers.telephony	ef1e345589
oxygen.agent	oxygen.agent	C:\data\data\oxygen.agent	0dc44f7c910
org.cyanogenmod.voiceplus	org.cyanogenmod.voiceplus	C:\data\data\org.cyanogenmod.voiceplus	3309176f1d
org.codeaurora.bluetooth	org.codeaurora.bluetooth	C:\data\data\org.codeaurora.bluetooth	b420d21926
Opera mini	com.opera.mini.android	C:\data\data\com.opera.mini.android	0c4f2c5271
net.zedge.android	net.zedge.android	C:\sdcard\Android\data\net.zedge.android	5b6a356ebd
Media	com.android.providers.media	C:\data\data\com.android.providers.media	f688d2cba9
lost+found	lost+found	C:\data\data\lost+found	fd6edeaa5f
jackpal.androidterm	jackpal.androidterm	C:\data\data\jackpal.androidterm	ebe4bd5e37
Google+	com.google.android.apps.plus	C:\data\data\com.google.android.apps.plus	525daa2dba
Google Talk	GoogleTalk	N/A	c40730d304
Google Maps	com.google.android.apps.maps	C:\sdcard\Android\data\com.google.android.apps.maps	f516dbb160
Google Mail	Google Mail	N/A	86e332213e
Google Chrome	com.android.chrome	C:\data\data\com.android.chrome	1a44c31756
Google Calendar	com.android.providers.calendar	C:\data\data\com.android.providers.calendar	ebab1b3d7c
Firefox Browser	org.mozilla.firefox	C:\data\data\org.mozilla.firefox	14f952e484
Facebook	com.facebook.katana	C:\data\data\com.facebook.katana	2e56b1e1f0
E-mail	com.android.email	C:\data\data\com.android.email	8fa3b5d268
Downloads	com.android.providers.downloads	C:\data\data\com.android.providers.downloads	d43177d6b8
Dolphin Browser HD	mobi.mgeek.TunnyBrowser	C:\data\data\mobi.mgeek.TunnyBrowser	29c47c6a40
com.wuala.android	com.wuala.android	C:\sdcard\Android\data\com.wuala.android	318e830d50
com.ubuntuone.android.files	com.ubuntuone.android.files	C:\sdcard\Android\data\com.ubuntuone.android.files	53ff995cc7f

E-mail · User data (3) · Application files (9)

Copy to clipboard · Viewer panel · Show thumbnails · Autosize · Export to Google Earth

Accounts	Display name	Sender name	E-mail address
	testing2086@gmail.com	Test	testing2086@gmail.com

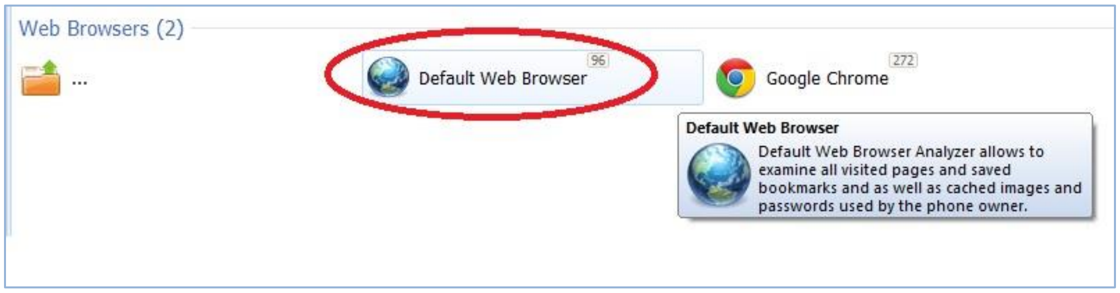
Passwords	Login	Password	Encrypted	Mail server URL	Protocol	Port
	testing2086@gmail.com	KALIMERA_12345	No	http:\\imap.gmail.com	imap	993
	testing2086@gmail.com	KALIMERA_12345	No	http:\\smtp.gmail.com	smtp	465

Categories
All categories (3)
Accounts (1)
Passwords (2)

Figure 106

5.2.2.1.6.2) Web browsers

- Regarding the browser, Oxygen Forensic Suite tool could regain data from the 2 browsers (default Web browser and Google Chrome):

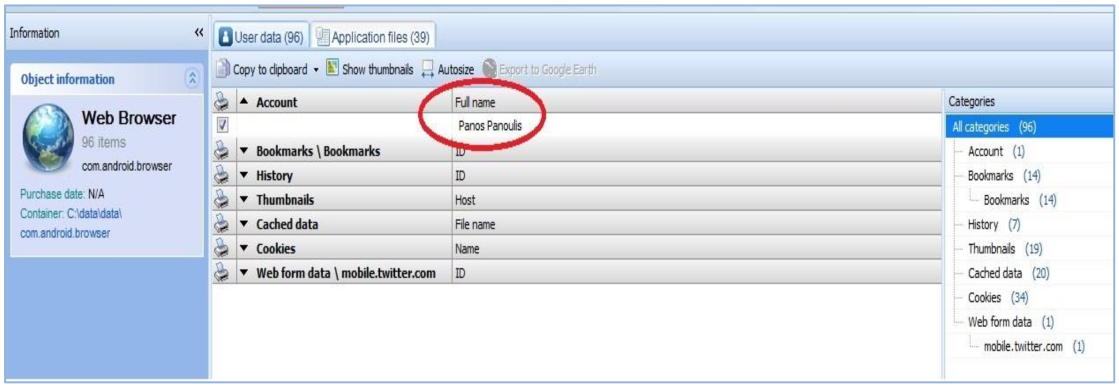


Default Web Browser

Default Web Browser Analyzer allows to examine all visited pages and saved bookmarks and as well as cached images and passwords used by the phone owner.

Figure 107

- In particularly, the default Web browser only manifested in full name and the email address of the user:

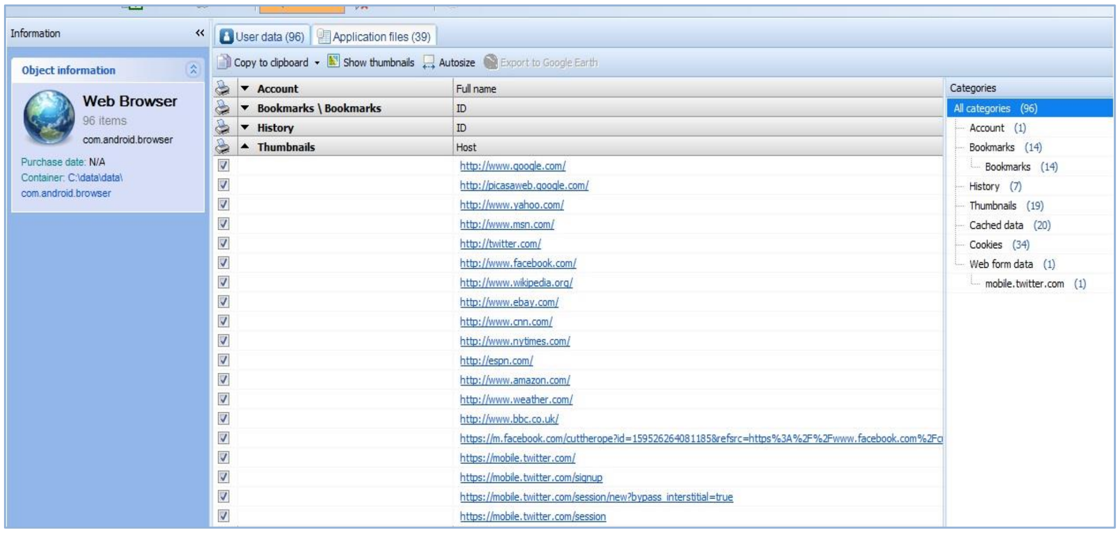


Web Browser
96 items
com.android.browser
Purchase date: N/A
Container: C:\data\data\com.android.browser

Account	Full name
	Panos Panoulis
Bookmarks \ Bookmarks	ID
History	ID
Thumbnails	Host
Cached data	File name
Cookies	Name
Web form data \ mobile.twitter.com	ID

Categories
All categories (96)
Account (1)
Bookmarks (14)
Bookmarks (14)
History (7)
Thumbnails (19)
Cached data (20)
Cookies (34)
Web form data (1)
mobile.twitter.com (1)

Figure 108

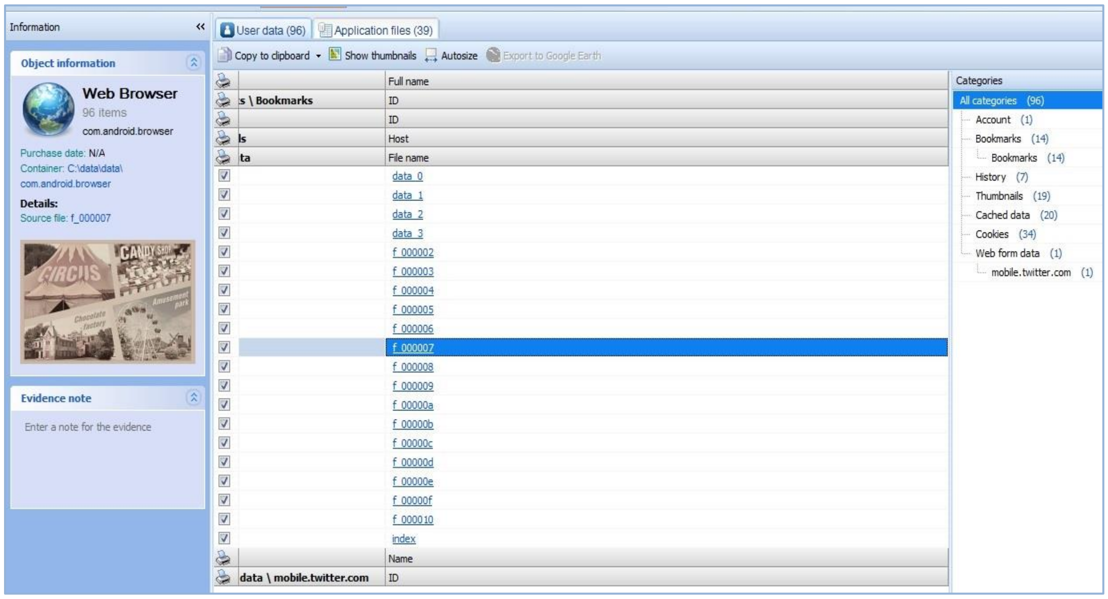


Web Browser
96 items
com.android.browser
Purchase date: N/A
Container: C:\data\data\com.android.browser

Account	Full name
Bookmarks \ Bookmarks	ID
History	ID
Thumbnails	Host
	http://www.google.com/
	http://picasaweb.google.com/
	http://www.yahoo.com/
	http://www.msn.com/
	http://twitter.com/
	http://www.facebook.com/
	http://www.wikipedia.org/
	http://www.ebay.com/
	http://www.cnn.com/
	http://www.nytimes.com/
	http://espn.com/
	http://www.amazon.com/
	http://www.weather.com/
	http://www.bbc.co.uk/
	https://m.facebook.com/cuttherope?id=15952626408118&refsrc=https%3A%2F%2Fwww.facebook.com%2Fc
	https://mobile.twitter.com/
	https://mobile.twitter.com/signup
	https://mobile.twitter.com/session/new?bypass_interstitial=true
	https://mobile.twitter.com/session

Categories
All categories (96)
Account (1)
Bookmarks (14)
Bookmarks (14)
History (7)
Thumbnails (19)
Cached data (20)
Cookies (34)
Web form data (1)
mobile.twitter.com (1)

Figure 109



Information

Object information

Web Browser
96 items
com.android.browser

Purchase date: N/A
Container: C:\data\data\com.android.browser

Details:
Source file: f_000007

Evidence note
Enter a note for the evidence

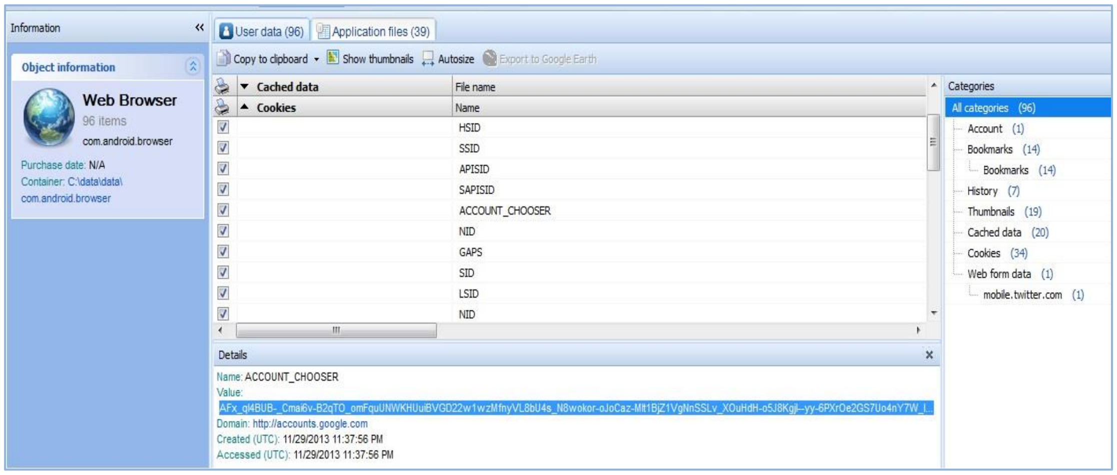
User data (96) | Application files (39)

Copy to clipboard | Show thumbnails | Autosize | Export to Google Earth

		Full name
	s \ Bookmarks	ID
		ID
	ls	Host
	ta	File name
✓		data_0
✓		data_1
✓		data_2
✓		data_3
✓		f_000002
✓		f_000003
✓		f_000004
✓		f_000005
✓		f_000006
✓		f_000007
✓		f_000008
✓		f_000009
✓		f_00000a
✓		f_00000b
✓		f_00000c
✓		f_00000d
✓		f_00000e
✓		f_00000f
✓		f_000010
✓		index
		Name
	data \ mobile.twitter.com	ID

Categories
- All categories (96)
  - Account (1)
  - Bookmarks (14)
    - Bookmarks (14)
  - History (7)
  - Thumbnails (19)
  - Cached data (20)
  - Cookies (34)
  - Web form data (1)
    - mobile.twitter.com (1)

Figure 110



Information

Object information

Web Browser
96 items
com.android.browser

Purchase date: N/A
Container: C:\data\data\com.android.browser

User data (96) | Application files (39)

Copy to clipboard | Show thumbnails | Autosize | Export to Google Earth

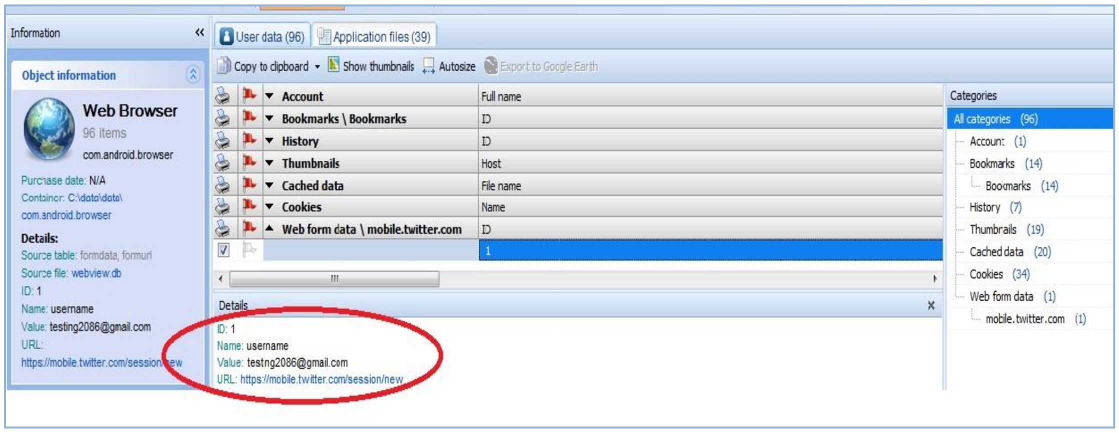
	▼ Cached data	File name
	▲ Cookies	Name
✓		HSID
✓		SSID
✓		APISID
✓		SAPISID
✓		ACCOUNT_CHOOSER
✓		NID
✓		GAPS
✓		SID
✓		LSID
✓		NID

Details

Name: ACCOUNT_CHOOSER
Value:
AFx_ql4BUB-_Cmai6v-B2qTO_omFquUNWKHUuiBVGD22w1wzMfnyVL8bU4s_N8wokor-oJoCaz-Mlt1BjZ1VgNnSSLv_XOuHdH-o5J8Kgjl--yy-6PXrOe2GS7Uo4nY7W_l...
Domain: http://accounts.google.com
Created (UTC): 11/29/2013 11:37:56 PM
Accessed (UTC): 11/29/2013 11:37:56 PM

Categories
- All categories (96)
  - Account (1)
  - Bookmarks (14)
    - Bookmarks (14)
  - History (7)
  - Thumbnails (19)
  - Cached data (20)
  - Cookies (34)
  - Web form data (1)
    - mobile.twitter.com (1)

Figure 111



Web Browser
96 items
com.android.browser
Purchase date: N/A
Container: C:\data\data\com.android.browser
Details:
Source table: formdata, formurl
Source file: webview.db
ID: 1
Name: username
Value: testing2086@gmail.com
URL: https://mobile.twitter.com/session/new

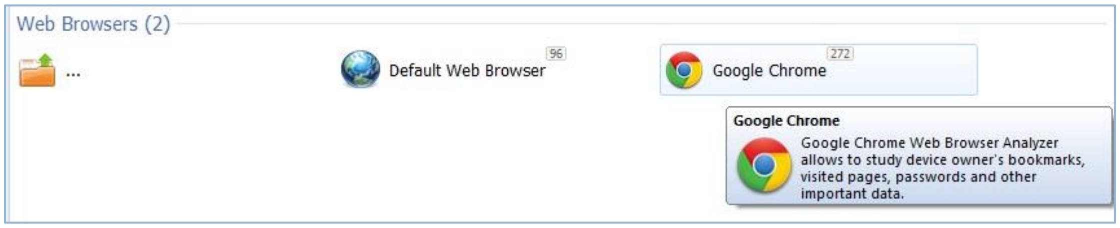
Account	Full name
Bookmarks \ Bookmarks	ID
History	ID
Thumbnails	Host
Cached data	File name
Cookies	Name
Web form data \ mobile.twitter.com	ID
	1

Details
ID: 1
Name: username
Value: testng2086@gmail.com
URL: https://mobile.twitter.com/session/new

Categories
All categories (96)
Account (1)
Bookmarks (14)
Bookmarks (14)
History (7)
Thumbnails (19)
Cached data (20)
Cookies (34)
Web form data (1)
mobile.twitter.com (1)

Figure 112

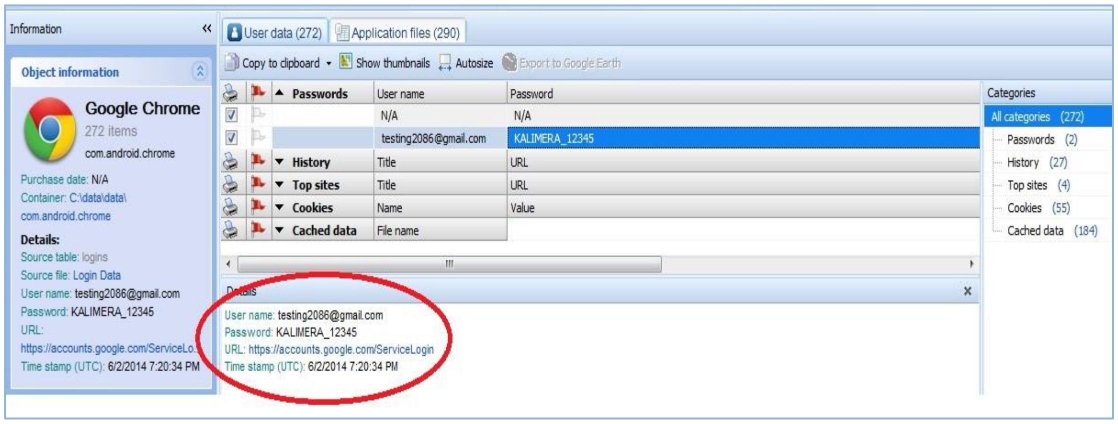
- The second case concerns the Google Chrome browser:



Google Chrome
Google Chrome Web Browser Analyzer allows to study device owner's bookmarks, visited pages, passwords and other important data.

Figure 113

- The tool could revealed the credentials (username and password) of the user during the navigation in the browser:



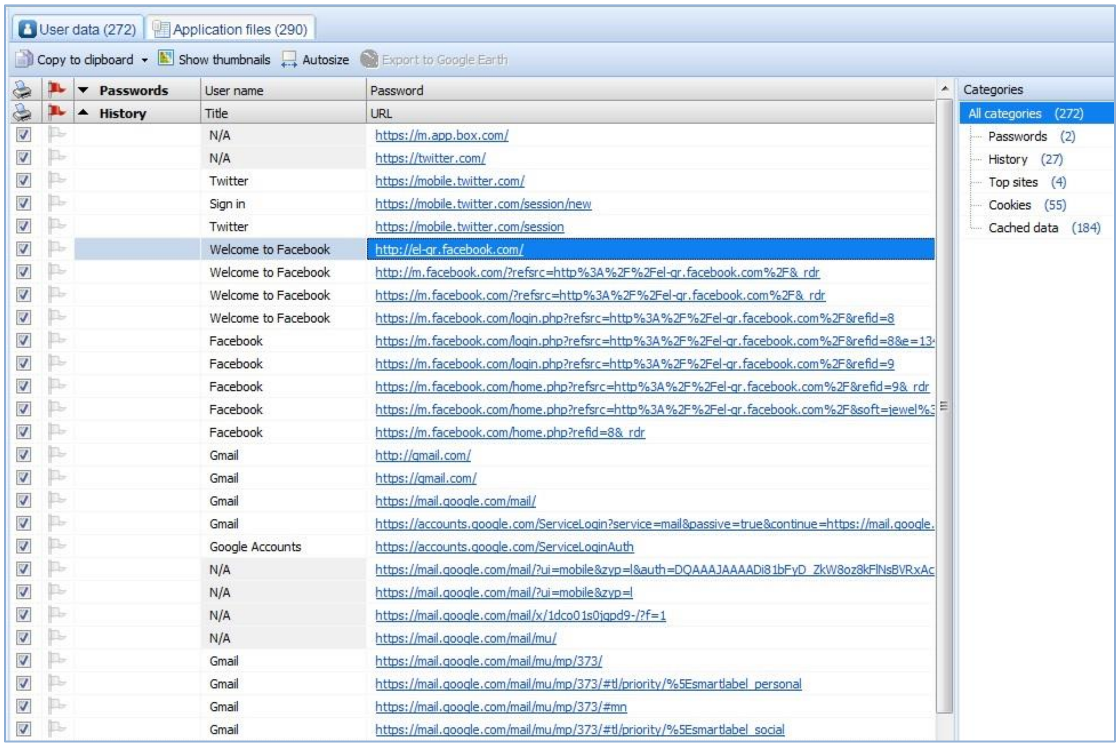
Google Chrome
272 items
com.android.chrome
Purchase date: N/A
Container: C:\data\data\com.android.chrome
Details:
Source table: logins
Source file: Login Data
User name: testing2086@gmail.com
Password: KALIMERA_12345
URL: https://accounts.google.com/ServiceLo...
Time stamp (UTC): 6/2/2014 7:20:34 PM

Passwords	User name	Password
	N/A	N/A
	testing2086@gmail.com	KALIMERA_12345
History	Title	URL
Top sites	Title	URL
Cookies	Name	Value
Cached data	File name	

Details
User name: testing2086@gmail.com
Password: KALIMERA_12345
URL: https://accounts.google.com/ServiceLogin
Time stamp (UTC): 6/2/2014 7:20:34 PM

Categories
All categories (272)
Passwords (2)
History (27)
Top sites (4)
Cookies (55)
Cached data (184)

Figure 114



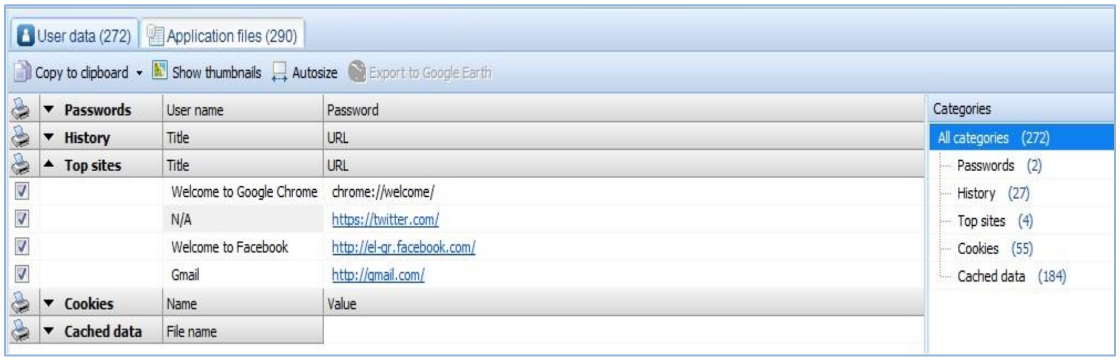
User data (272) | Application files (290)

Copy to clipboard | Show thumbnails | Autosize | Export to Google Earth

▼ Passwords	User name	Password	
▲ History	Title	URL	
✓	N/A	https://m.app.box.com/	
✓	N/A	https://twitter.com/	
✓	Twitter	https://mobile.twitter.com/	
✓	Sign in	https://mobile.twitter.com/session/new	
✓	Twitter	https://mobile.twitter.com/session	
✓	Welcome to Facebook	http://el-gr.facebook.com/	
✓	Welcome to Facebook	http://m.facebook.com/?refsrc=http%3A%2F%2Fel-gr.facebook.com%2F&_rdr	
✓	Welcome to Facebook	https://m.facebook.com/?refsrc=http%3A%2F%2Fel-gr.facebook.com%2F&_rdr	
✓	Welcome to Facebook	https://m.facebook.com/login.php?refsrc=http%3A%2F%2Fel-gr.facebook.com%2F&refid=8	
✓	Facebook	https://m.facebook.com/login.php?refsrc=http%3A%2F%2Fel-gr.facebook.com%2F&refid=8&e=13	
✓	Facebook	https://m.facebook.com/login.php?refsrc=http%3A%2F%2Fel-gr.facebook.com%2F&refid=9	
✓	Facebook	https://m.facebook.com/home.php?refsrc=http%3A%2F%2Fel-gr.facebook.com%2F&refid=9&_rdr	
✓	Facebook	https://m.facebook.com/home.php?refsrc=http%3A%2F%2Fel-gr.facebook.com%2F&soft=jewel%3	
✓	Facebook	https://m.facebook.com/home.php?refid=8&_rdr	
✓	Gmail	http://gmail.com/	
✓	Gmail	https://gmail.com/	
✓	Gmail	https://mail.google.com/mail/	
✓	Gmail	https://accounts.google.com/ServiceLogin?service=mail&passive=true&continue=https://mail.google.	
✓	Google Accounts	https://accounts.google.com/ServiceLoginAuth	
✓	N/A	https://mail.google.com/mail/?ui=mobile&zyp=l&auth=DQAAAJAAAADi81bFyD_ZkW8oz8kFlNsBVRxAc	
✓	N/A	https://mail.google.com/mail/?ui=mobile&zyp=l	
✓	N/A	https://mail.google.com/mail/x/1dco01s0jgpd9-/?f=1	
✓	N/A	https://mail.google.com/mail/mu/	
✓	Gmail	https://mail.google.com/mail/mu/mp/373/	
✓	Gmail	https://mail.google.com/mail/mu/mp/373/#tl/priority/%5Esmartlabel_personal	
✓	Gmail	https://mail.google.com/mail/mu/mp/373/#mn	
✓	Gmail	https://mail.google.com/mail/mu/mp/373/#tl/priority/%5Esmartlabel_social	

Categories
All categories (272)
Passwords (2)
History (27)
Top sites (4)
Cookies (55)
Cached data (184)

Figure 115



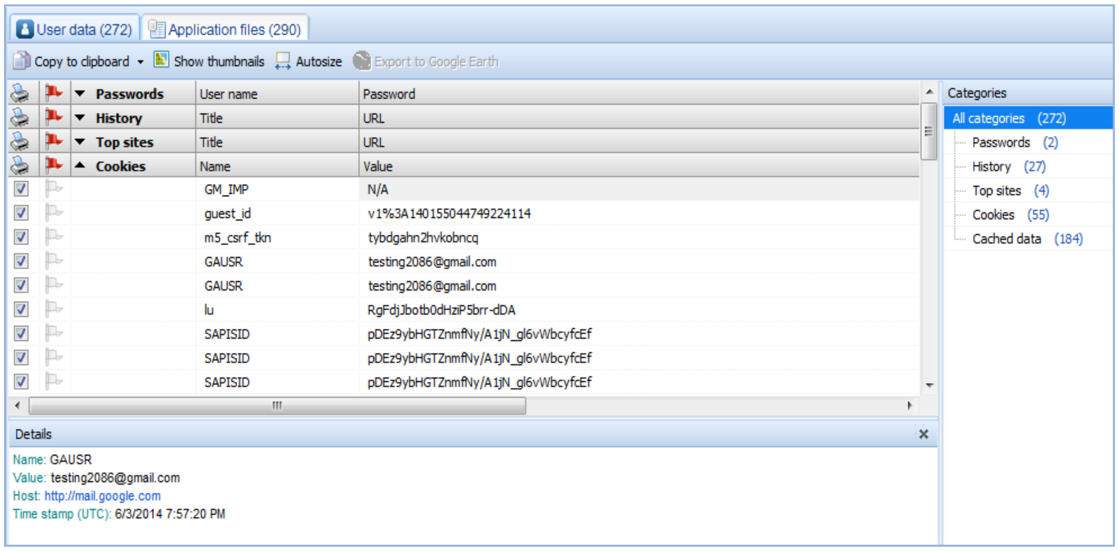
User data (272) | Application files (290)

Copy to clipboard | Show thumbnails | Autosize | Export to Google Earth

▼ Passwords	User name	Password
▼ History	Title	URL
▲ Top sites	Title	URL
✓	Welcome to Google Chrome	chrome://welcome/
✓	N/A	https://twitter.com/
✓	Welcome to Facebook	http://el-gr.facebook.com/
✓	Gmail	http://gmail.com/
▼ Cookies	Name	Value
▼ Cached data	File name	

Categories
All categories (272)
Passwords (2)
History (27)
Top sites (4)
Cookies (55)
Cached data (184)

Figure 116



Passwords User name	Password
History Title	URL
Top sites Title	URL
Cookies Name	Value
GM_IMP	N/A
guest_id	v1%3A140155044749224114
m5_csrf_tkn	tybdgahn2hvkobncq
GAUSR	testing2086@gmail.com
GAUSR	testing2086@gmail.com
lu	RgFdjJbotb0dHziP5brr-dDA
SAPISID	pDEz9ybHGTZnmfNy/A1jN_gl6vWbcyfcEf
SAPISID	pDEz9ybHGTZnmfNy/A1jN_gl6vWbcyfcEf
SAPISID	pDEz9ybHGTZnmfNy/A1jN_gl6vWbcyfcEf

Details
Name: GAUSR
Value: testing2086@gmail.com
Host: http://mail.google.com
Time stamp (UTC): 6/3/2014 7:57:20 PM

Figure 117

5.2.2.1.6.3) File browser

File Browser section is a powerful tool to access and analyze user photos, videos, documents and device databases.

Built-in text, hex, multimedia, SQLite, Plist viewers, Geo-location and EXIF extractors help experts to view files and their properties.

File Browser gives the access to all files extracted from the device in a tree-based structure. Additionally experts can view files in selected only folders, grouped in tabs by type, or in search tabs created when Quick filter was applied.

Oxygen Forensic Suite automatically extracts Geo-location data from multimedia files and offers shortcut to view the place on the map where the action took place. Additionally, it looks into EXIF headers for specific tags like Make, Model, time stamps which helps forensic experts to determine file origin. Basic data about the file is displayed on the left sidebar, while file attributes and additional tags are available via popup menu.

Section has a two-panel multipurpose viewer, where experts view the file in a raw, hex mode, or run appropriate player for the media file. Additionally, double clicking on a file opens appropriate viewer in a separate window. Databases and configuration files will be opened with built-in Oxygen Forensic SQLite Viewer and Oxygen Forensic Plist Viewer.

Investigators can save files, whole folders on PC, mark the items as Key Evidence, print and prepare reports, export Geo data to Google Earth.

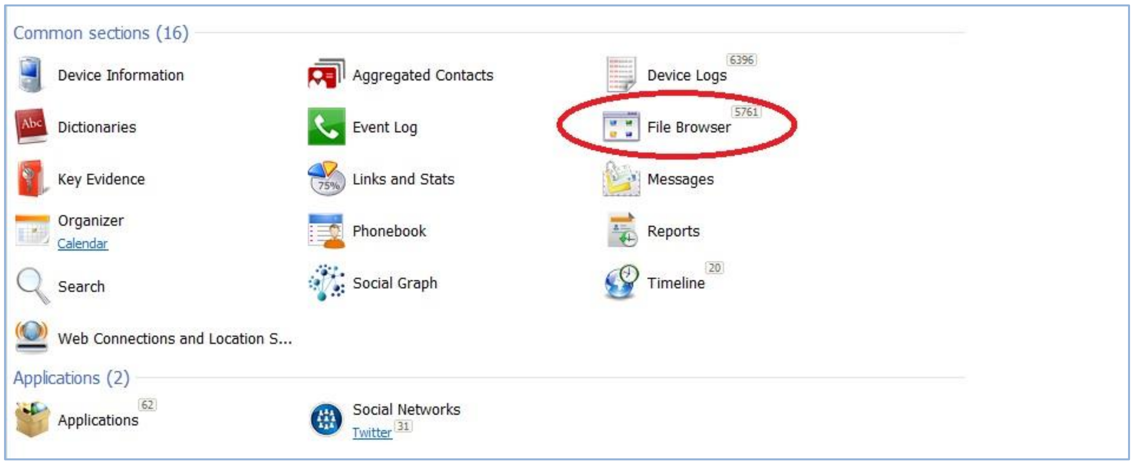
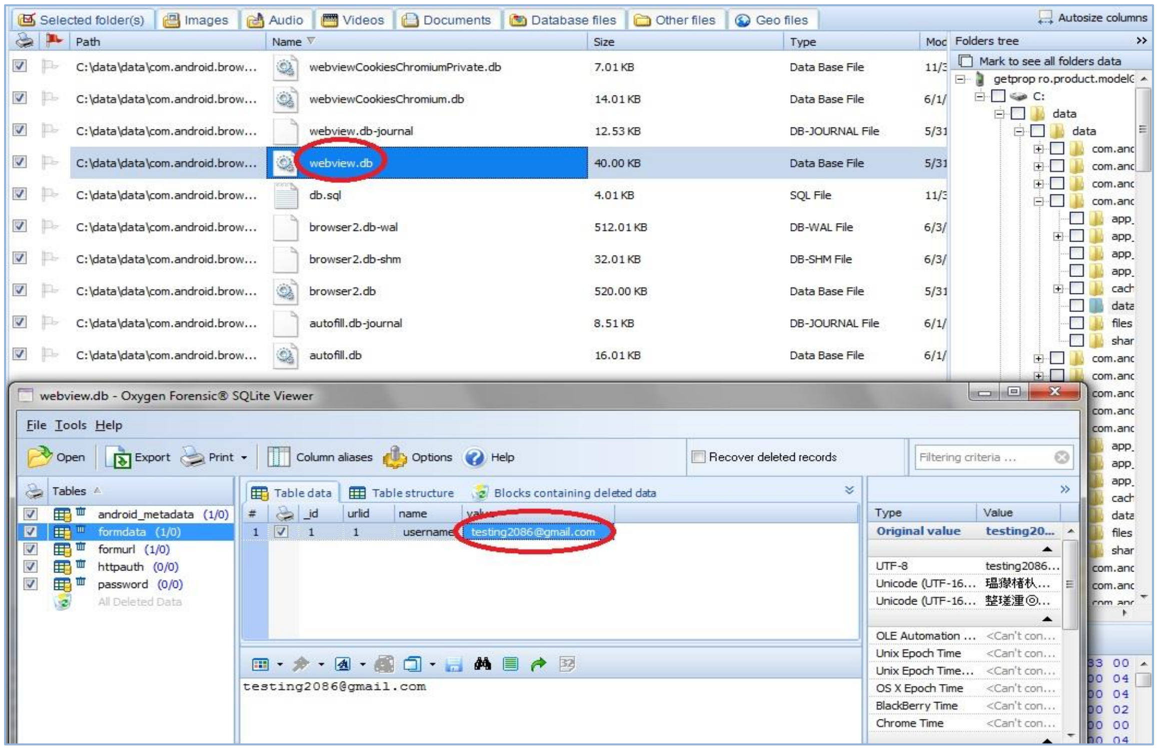


Figure 118

File browser allows through the paths (like file manager) to navigate to the mobile device applications and their databases to retrieve information.

- Starting with the first application (the default Web browser) and the database “webview.db” we can find the user's email:



Path	Name	Size	Type
C:\data\data\com.android.brow...	webviewCookiesChromiumPrivate.db	7.01 KB	Data Base File
C:\data\data\com.android.brow...	webviewCookiesChromium.db	14.01 KB	Data Base File
C:\data\data\com.android.brow...	webview.db-journal	12.53 KB	DB-JOURNAL File
C:\data\data\com.android.brow...	webview.db	40.00 KB	Data Base File
C:\data\data\com.android.brow...	db.sql	4.01 KB	SQL File
C:\data\data\com.android.brow...	browser2.db-wal	512.01 KB	DB-WAL File
C:\data\data\com.android.brow...	browser2.db-shm	32.01 KB	DB-SHM File
C:\data\data\com.android.brow...	browser2.db	520.00 KB	Data Base File
C:\data\data\com.android.brow...	autofill.db-journal	8.51 KB	DB-JOURNAL File
C:\data\data\com.android.brow...	autofill.db	16.01 KB	Data Base File

Tables
android_metadata (1/0)
formdata (1/0)
formurl (1/0)
httpauth (0/0)
password (0/0)
All Deleted Data

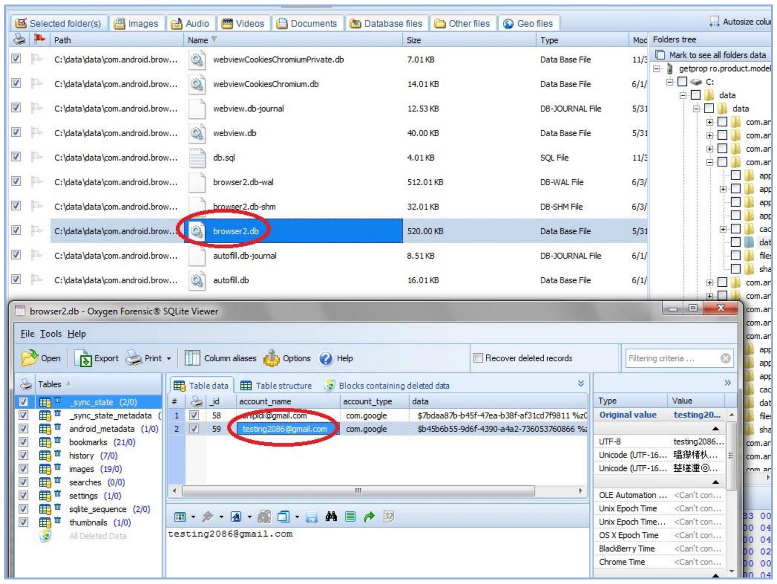
#	_id	urlid	name	value
1	1	1	username	testing2086@gmail.com

```
testing2086@gmail.com
```

Type	Value
Original value	testing20...
UTF-8	testing2086...
Unicode (UTF-16...	瑥瑳湩㉧...
Unicode (UTF-16...	整瑳湩...
OLE Automation ...	<Can't con...
Unix Epoch Time	<Can't con...
Unix Epoch Time...	<Can't con...
OS X Epoch Time	<Can't con...
BlackBerry Time	<Can't con...
Chrome Time	<Can't con...

Figure 119

- Again with the same application (default Web browser) but with another databse “browser2.db” we face again the email address of the user:



Path	Name	Size	Type	Moc
C:\data\data\com.android.brow...	webviewCookiesChromiumPrivate.db	7.01 KB	Data Base File	11/3
C:\data\data\com.android.brow...	webviewCookiesChromium.db	14.01 KB	Data Base File	6/1/
C:\data\data\com.android.brow...	webview.db-journal	12.53 KB	DB-JOURNAL File	5/31
C:\data\data\com.android.brow...	webview.db	40.00 KB	Data Base File	5/31
C:\data\data\com.android.brow...	db.sql	4.01 KB	SQL File	11/3
C:\data\data\com.android.brow...	browser2.db-wal	512.01 KB	DB-WAL File	6/3/
C:\data\data\com.android.brow...	browser2.db-shm	32.01 KB	DB-SHM File	6/3/
C:\data\data\com.android.brow...	browser2.db	520.00 KB	Data Base File	5/31
C:\data\data\com.android.brow...	autofill.db-journal	8.51 KB	DB-JOURNAL File	6/1/
C:\data\data\com.android.brow...	autofill.db	16.01 KB	Data Base File	6/1/

browser2.db - Oxygen Forensic® SQLite Viewer

Tables: _sync_state (2/0), _sync_state_metadata, android_metadata (1/0), bookmarks (21/0), history (7/0), images (19/0), searches (0/0), settings (1/0), sqlite_sequence (2/0), thumbnails (1/0), All Deleted Data

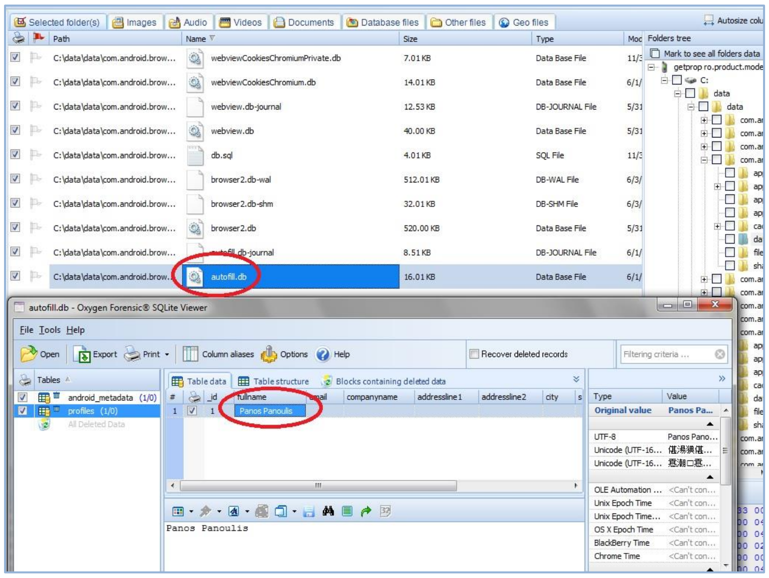
#	_id	account_name	account_type	data
1	58	[illegible]pidi@gmail.com	com.google	$7bdaa87b-b45f-47ea-b38f-af31cd7f9811 %z0
2	59	testing2086@gmail.com	com.google	$b45b6b55-9d6f-4390-a4a2-736053760866 %:

```
testing2086@gmail.com
```

Type	Value
Original value	testing20...
UTF-8	testing2086...
Unicode (UTF-16...	瑥獴楮朲...
Unicode (UTF-16...	整瑳湩ⓖ...
OLE Automation ...	<Can't con...
Unix Epoch Time	<Can't con...
Unix Epoch Time...	<Can't con...
OS X Epoch Time	<Can't con...
BlackBerry Time	<Can't con...
Chrome Time	<Can't con...

Figure 120

- The same result (full name) we have with the database “autofill.db”:



Path	Name	Size	Type	Moc
C:\data\data\com.android.brow...	webviewCookiesChromiumPrivate.db	7.01 KB	Data Base File	11/3
C:\data\data\com.android.brow...	webviewCookiesChromium.db	14.01 KB	Data Base File	6/1/
C:\data\data\com.android.brow...	webview.db-journal	12.53 KB	DB-JOURNAL File	5/31
C:\data\data\com.android.brow...	webview.db	40.00 KB	Data Base File	5/31
C:\data\data\com.android.brow...	db.sql	4.01 KB	SQL File	11/3
C:\data\data\com.android.brow...	browser2.db-wal	512.01 KB	DB-WAL File	6/3/
C:\data\data\com.android.brow...	browser2.db-shm	32.01 KB	DB-SHM File	6/3/
C:\data\data\com.android.brow...	browser2.db	520.00 KB	Data Base File	5/31
C:\data\data\com.android.brow...	autofill.db-journal	8.51 KB	DB-JOURNAL File	6/1/
C:\data\data\com.android.brow...	autofill.db	16.01 KB	Data Base File	6/1/

autofill.db - Oxygen Forensic® SQLite Viewer

Tables: android_metadata (1/0), profiles (1/0), All Deleted Data

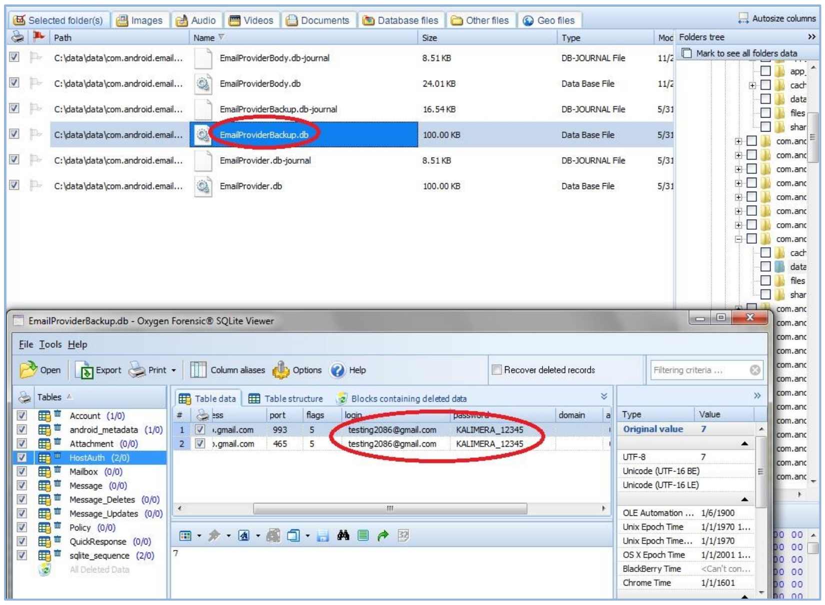
#	_id	fullname	mail	companyname	addressline1	addressline2	city	s
1	1	Panos Panoulis						

```
Panos Panoulis
```

Type	Value
Original value	Panos Pa...
UTF-8	Panos Pano...
Unicode (UTF-16...	倠湡獯倠...
Unicode (UTF-16...	慐潮□慐...
OLE Automation ...	<Can't con...
Unix Epoch Time	<Can't con...
Unix Epoch Time...	<Can't con...
OS X Epoch Time	<Can't con...
BlackBerry Time	<Can't con...
Chrome Time	<Can't con...

Figure 121

- From the application default Android email and the database "Email Provider Backup.db" the credential (username and password) of the user can be recovered:



Path	Name	Size	Type	Moc
C:\data\data\com.android.email...	EmailProviderBody.db-journal	8.51 KB	DB-JOURNAL File	11/2
C:\data\data\com.android.email...	EmailProviderBody.db	24.01 KB	Data Base File	11/2
C:\data\data\com.android.email...	EmailProviderBackup.db-journal	16.54 KB	DB-JOURNAL File	5/31
C:\data\data\com.android.email...	EmailProviderBackup.db	100.00 KB	Data Base File	5/31
C:\data\data\com.android.email...	EmailProvider.db-journal	8.51 KB	DB-JOURNAL File	5/31
C:\data\data\com.android.email...	EmailProvider.db	100.00 KB	Data Base File	5/31

EmailProviderBackup.db - Oxygen Forensic® SQLite Viewer

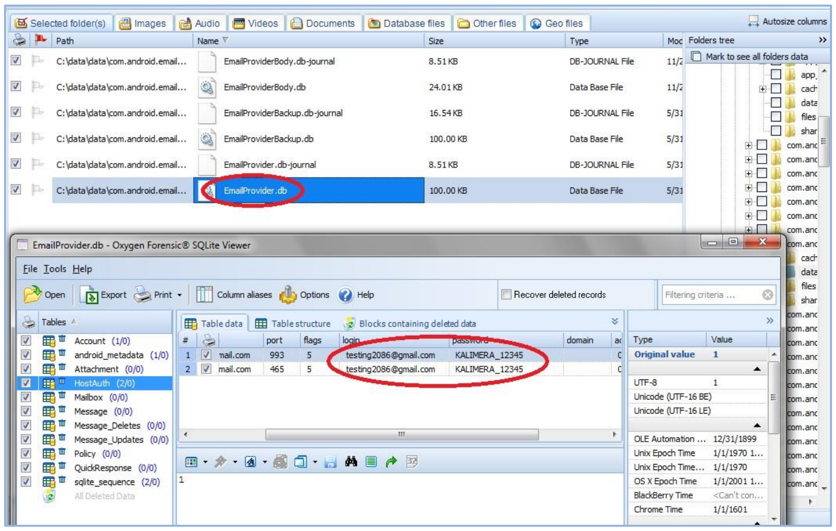
Tables: Account (1/0), android_metadata (1/0), Attachment (0/0), HostAuth (2/0), Mailbox (0/0), Message (0/0), Message_Deletes (0/0), Message_Updates (0/0), Policy (0/0), QuickResponse (0/0), sqlite_sequence (2/0), All Deleted Data

#	ess	port	flags	login	password	domain
1	ı.gmail.com	993	5	testing2086@gmail.com	KALIMERA_12345	
2	›.gmail.com	465	5	testing2086@gmail.com	KALIMERA_12345	

Type	Value
Original value	7
UTF-8	7
Unicode (UTF-16 BE)	
Unicode (UTF-16 LE)	
OLE Automation ...	1/6/1900
Unix Epoch Time	1/1/1970 1...
Unix Epoch Time...	1/1/1970
OS X Epoch Time	1/1/2001 1...
BlackBerry Time	<Can't con...
Chrome Time	1/1/1601

Figure 122

- From the same application (default Android email) but another database "EmailProvider.db" we can find the credentials of the user (username and password):



Path	Name	Size	Type	Moc
C:\data\data\com.android.email...	EmailProviderBody.db-journal	8.51 KB	DB-JOURNAL File	11/2
C:\data\data\com.android.email...	EmailProviderBody.db	24.01 KB	Data Base File	11/2
C:\data\data\com.android.email...	EmailProviderBackup.db-journal	16.54 KB	DB-JOURNAL File	5/31
C:\data\data\com.android.email...	EmailProviderBackup.db	100.00 KB	Data Base File	5/31
C:\data\data\com.android.email...	EmailProvider.db-journal	8.51 KB	DB-JOURNAL File	5/31
C:\data\data\com.android.email...	EmailProvider.db	100.00 KB	Data Base File	5/31

EmailProvider.db - Oxygen Forensic® SQLite Viewer

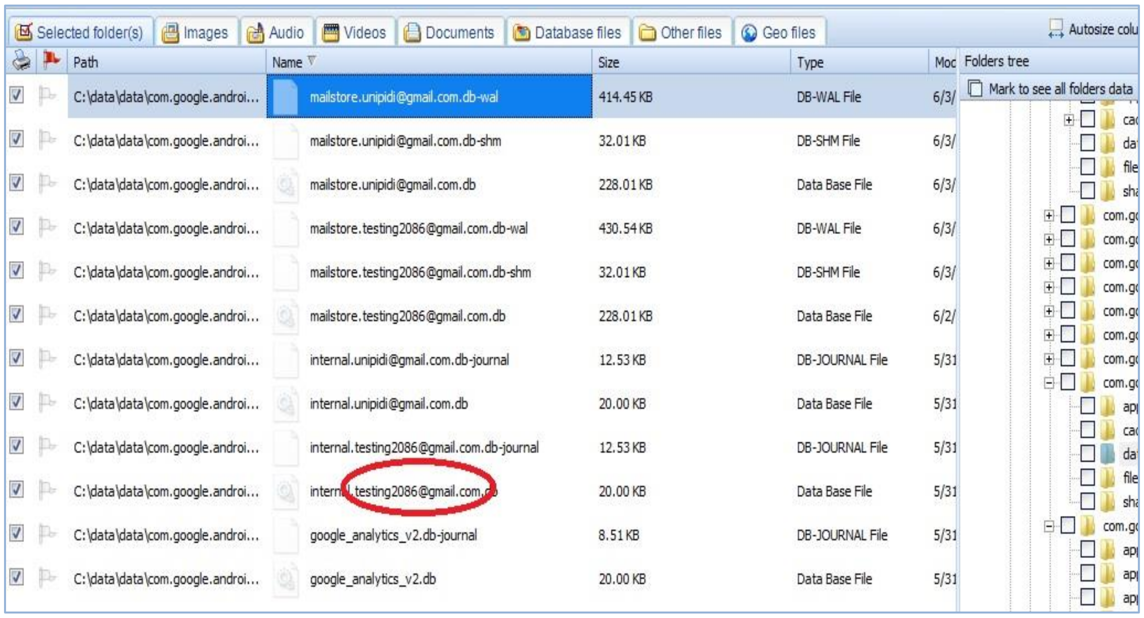
Tables: Account (1/0), android_metadata (1/0), Attachment (0/0), HostAuth (2/0), Mailbox (0/0), Message (0/0), Message_Deletes (0/0), Message_Updates (0/0), Policy (0/0), QuickResponse (0/0), sqlite_sequence (2/0), All Deleted Data

#		port	flags	login	password	domain
1	nail.com	993	5	testing2086@gmail.com	KALIMERA_12345	
2	mail.com	465	5	testing2086@gmail.com	KALIMERA_12345	

Type	Value
Original value	1
UTF-8	1
Unicode (UTF-16 BE)	
Unicode (UTF-16 LE)	
OLE Automation ...	12/31/1899
Unix Epoch Time	1/1/1970 1...
Unix Epoch Time...	1/1/1970
OS X Epoch Time	1/1/2001 1...
BlackBerry Time	<Can't con...
Chrome Time	1/1/1601

Figure 123

- Finishing, from the application email (Gmail), we can grab only the email address which is used by the user:

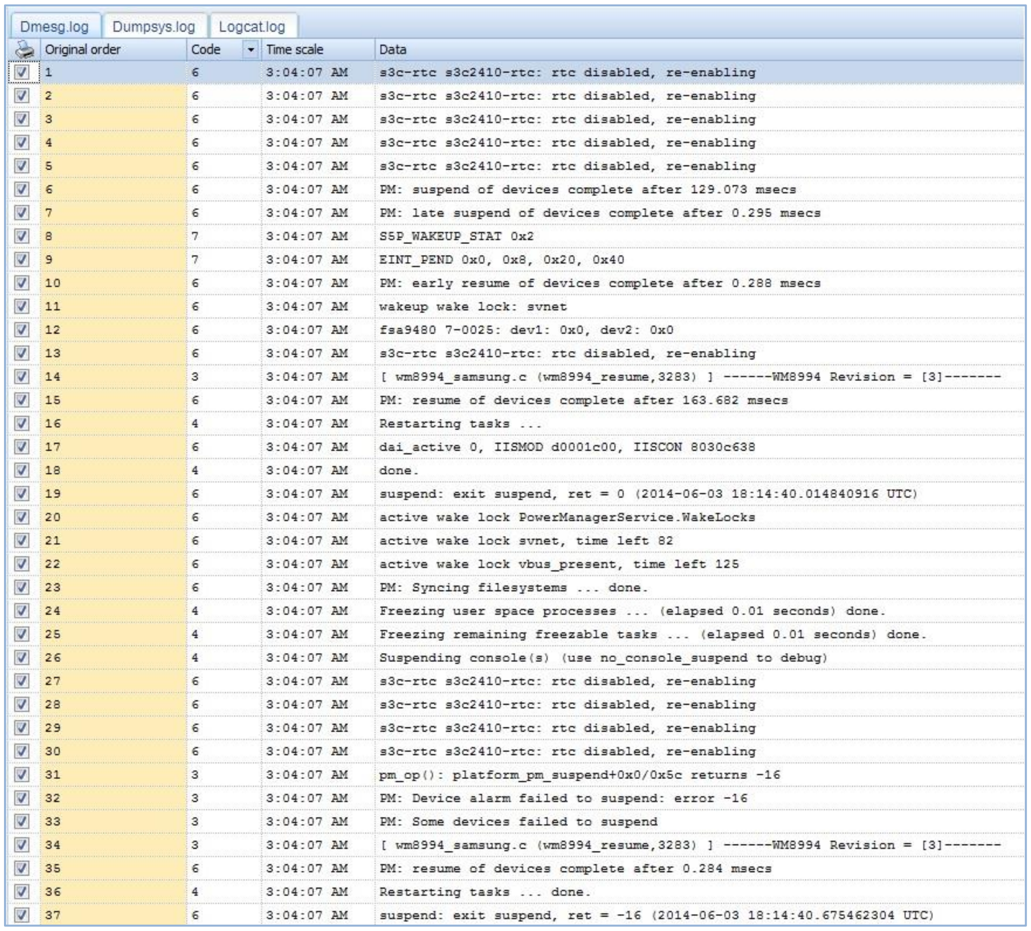


Path	Name	Size	Type	Moc
C:\data\data\com.google.androi...	mailstore.unipidi@gmail.com.db-wal	414.45 KB	DB-WAL File	6/3/
C:\data\data\com.google.androi...	mailstore.unipidi@gmail.com.db-shm	32.01 KB	DB-SHM File	6/3/
C:\data\data\com.google.androi...	mailstore.unipidi@gmail.com.db	228.01 KB	Data Base File	6/3/
C:\data\data\com.google.androi...	mailstore.testing2086@gmail.com.db-wal	430.54 KB	DB-WAL File	6/3/
C:\data\data\com.google.androi...	mailstore.testing2086@gmail.com.db-shm	32.01 KB	DB-SHM File	6/3/
C:\data\data\com.google.androi...	mailstore.testing2086@gmail.com.db	228.01 KB	Data Base File	6/2/
C:\data\data\com.google.androi...	internal.unipidi@gmail.com.db-journal	12.53 KB	DB-JOURNAL File	5/31
C:\data\data\com.google.androi...	internal.unipidi@gmail.com.db	20.00 KB	Data Base File	5/31
C:\data\data\com.google.androi...	internal.testing2086@gmail.com.db-journal	12.53 KB	DB-JOURNAL File	5/31
C:\data\data\com.google.androi...	internal.testing2086@gmail.com.db	20.00 KB	Data Base File	5/31
C:\data\data\com.google.androi...	google_analytics_v2.db-journal	8.51 KB	DB-JOURNAL File	5/31
C:\data\data\com.google.androi...	google_analytics_v2.db	20.00 KB	Data Base File	5/31

Figure 124

5.2.2.1.6.4) Device logs

- Oxygen Forensic Suite is able to recover intact throughout the history of movement of a mobile device from the initial time of operation:



	Original order	Code	Time scale	Data
✓	1	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	2	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	3	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	4	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	5	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	6	6	3:04:07 AM	PM: suspend of devices complete after 129.073 msecs
✓	7	6	3:04:07 AM	PM: late suspend of devices complete after 0.295 msecs
✓	8	7	3:04:07 AM	S5P_WAKEUP_STAT 0x2
✓	9	7	3:04:07 AM	EINT_PEND 0x0, 0x8, 0x20, 0x40
✓	10	6	3:04:07 AM	PM: early resume of devices complete after 0.288 msecs
✓	11	6	3:04:07 AM	wakeup wake lock: svnet
✓	12	6	3:04:07 AM	fsa9480 7-0025: dev1: 0x0, dev2: 0x0
✓	13	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	14	3	3:04:07 AM	[wm8994_samsung.c (wm8994_resume,3283)] ------WM8994 Revision = [3]-------
✓	15	6	3:04:07 AM	PM: resume of devices complete after 163.682 msecs
✓	16	4	3:04:07 AM	Restarting tasks ...
✓	17	6	3:04:07 AM	dai_active 0, IISMOD d0001c00, IISCON 8030c638
✓	18	4	3:04:07 AM	done.
✓	19	6	3:04:07 AM	suspend: exit suspend, ret = 0 (2014-06-03 18:14:40.014840916 UTC)
✓	20	6	3:04:07 AM	active wake lock PowerManagerService.WakeLocks
✓	21	6	3:04:07 AM	active wake lock svnet, time left 82
✓	22	6	3:04:07 AM	active wake lock vbus_present, time left 125
✓	23	6	3:04:07 AM	PM: Syncing filesystems ... done.
✓	24	4	3:04:07 AM	Freezing user space processes ... (elapsed 0.01 seconds) done.
✓	25	4	3:04:07 AM	Freezing remaining freezable tasks ... (elapsed 0.01 seconds) done.
✓	26	4	3:04:07 AM	Suspending console(s) (use no_console_suspend to debug)
✓	27	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	28	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	29	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	30	6	3:04:07 AM	s3c-rtc s3c2410-rtc: rtc disabled, re-enabling
✓	31	3	3:04:07 AM	pm_op(): platform_pm_suspend+0x0/0x5c returns -16
✓	32	3	3:04:07 AM	PM: Device alarm failed to suspend: error -16
✓	33	3	3:04:07 AM	PM: Some devices failed to suspend
✓	34	3	3:04:07 AM	[wm8994_samsung.c (wm8994_resume,3283)] ------WM8994 Revision = [3]-------
✓	35	6	3:04:07 AM	PM: resume of devices complete after 0.284 msecs
✓	36	4	3:04:07 AM	Restarting tasks ... done.
✓	37	6	3:04:07 AM	suspend: exit suspend, ret = -16 (2014-06-03 18:14:40.675462304 UTC)

Figure 125

5.2.2.1.7) Summary

Oxygen Forensic Suite 2014, based on results and performance, is considered a highly satisfactory data extraction tool Apart from the usual type of data: pictures, videos, contacts, messages, it can invade deep one the device and retrieve the credentials (e.g. usernames and passwords) from (mobile) applications. The methodology which Oxygen Forensic Suite followed is relatively simple as taking a first export the entire ROM memory (mobile) and then analyzes the data.

It was observed that from the only apps that the credentials (username and password) of the user were revealed, were type of Browser (e.g. Google Browser) and default e-mail (e.g. Default Android Email).

5.2.2.2) 3rd Scenario: Linux Memory Extractor Tool (LIME Forensics)

5.2.2.2.1) Compiling LiME

5.2.2.2.1.1) Compilation for Linux

Introduction

"kbuild" is the build system used by the Linux kernel. Modules must use kbuild to stay compatible with changes in the build infrastructure and to pick up the right flags to "gcc." Functionality for building modules both in tree and out of tree is provided. The method for building either is similar, and all modules are initially developed and built out of tree. Covered in this document is information aimed at developers interested in building out of tree (or "external") modules. The author of an external module should supply a makefile that hides most of the complexity, so one only has to type "make" to build the module.

How to Build External Modules

To build external modules, you must have a prebuilt kernel available that contains the configuration and header files used in the build. Also, the kernel must have been built with modules enabled. If you are using a distribution kernel, there will be a package for the kernel you are running provided by your distribution.

An alternative is to use the "make" target "modules_prepare." This will make sure the kernel contains the information required. The target exists solely as a simple way to prepare a kernel source tree for building external modules.

NOTE: "modules_prepare" will not build Module.symvers even if CONFIG_MODVERSIONS is set; therefore, a full kernel build needs to be executed to make module versioning work.

Command Syntax

The command to build an external module is:

```
$ make C <path_to_kernel_src> M=$PWD
```

The kbuild system knows that an external module is being built due to the "M=<dir>" option given in the command.

To build against the running kernel use:

```
$ make C /lib/modules/‘uname r‘/build M=$PWD
```

Then to install the module(s) just built, add the target "modules_install" to the command:

```
$ make C /lib/modules/‘uname r‘/build M=$PWD modules_install
```

Options

($KDIR refers to the path of the kernel source directory.)

make C $KDIR M=$PWD

C $KDIR

The directory where the kernel source is located. "make" will actually change to the specified directory 95 when executing and will change back when finished.

M=$PWD

Informs kbuild that an external module is being built. The value given to "M" is the absolute path of the 100 directory where the external module (kbuild file) is located.

Targets

When building an external module, only a subset of the "make" targets are available.

make C $KDIR M=$PWD [target]

The default will build the module(s) located in the current directory, so a target does not need to be specified. All output files will also be generated in this directory. No attempts are made to update the kernel source, and it is a precondition that a successful "make" has been executed for the kernel.

Modules

The default target for external modules. It has the same functionality as if no target was specified.

modules_install

Install the external module(s). The default location is /lib/modules/<kernel_release>/extra/, but a prefix may be added with INSTALL_MOD_PATH.

Clean

Remove all generated files in the module directory only.

Help

List the available targets for external modules.

Building Separate Files

It is possible to build single files that are part of a module. This works equally well for the kernel, a module, and even for external modules.

```
Example (The module foo.ko, consist of bar.o and baz.o):

make C $KDIR M=$PWD bar.lst
make C $KDIR M=$PWD baz.o
make C $KDIR M=$PWD foo.ko
make C $KDIR M=$PWD /
```

Creating a Kbuild File for an External Module

In the last section we saw the command to build a module for the running kernel. The module is not actually built, however, because a build file is required. Contained in this file will be the name of the module(s) being built, along with the list of requisite source files. The file may be as simple as a single line:

```
obj m := <module_name>.o
```

The kbuild system will build <module_name>.o from <module_name>.c, and, after linking, will result in the kernel module <module_name>.ko. The above line can be put in either a "Kbuild" file or a "Makefile." When the module is built from multiple sources, an additional line is needed listing the files:

```
<module_name> y := <src1>.o <src2>.o
```

NOTE: Further documentation describing the syntax used by kbuild is located in Documentation/kbuild/makefiles.txt.

The examples below demonstrate how to create a build file for the module 8123.ko, which is built from the following files:

```
8123_if.c
8123_if.h
8123_pci.c
8123_bin.o_shipped <= Binary blob
```

Shared Makefile

An external module always includes a wrapper makefile that supports building the module using "make" with no arguments. This target is not used by kbuild; it is only for convenience. Additional functionality, such as test targets, can be included but should be filtered out from kbuild due to possible name clashes.

```
Example 1:

> filename: Makefile
ifneq ($(KERNELRELEASE),)
# kbuild part of makefile
obj m := 8123.o
8123 y := 8123_if.o 8123_pci.o 8123_bin.o

else

# normal makefile
KDIR ?= /lib/modules/‘uname r‘/build
default:
$(MAKE) C $(KDIR) M=$$PWD
# Module specific targets
genbin:
echo "X" > 8123_bin.o_shipped
endif
```

The check for KERNELRELEASE is used to separate the two parts of the makefile. In the example, kbuild will only see the two assignments, whereas "make" will see everything except these two assignments. This is due to two passes made on the file: the first pass is by the "make" instance run on the command
line; the second pass is by the kbuild system, which is initiated by the parameterized "make" in the default target.

Separate Kbuild File and Makefile

In newer versions of the kernel, kbuild will first look for a file named "Kbuild," and only if that is not found, will it then look for a makefile. Utilizing a "Kbuild" file allows us to split up the makefile from example 1 into two files:

```
Example 2:

--> filename: Kbuild
obj m := 8123.o
8123 y := 8123_if.o 8123_pci.o 8123_bin.o

--> filename: Makefile
KDIR ?= /lib/modules/‘uname r‘/build
default:
```

```
$(MAKE) C $(KDIR) M=$$PWD
# Module specific targets
genbin:
echo "X" > 8123_bin.o_shipped
```

The split in example 2 is questionable due to the simplicity of each file; however, some external modules use makefiles consisting of several hundred lines, and here it really pays off to separate the kbuild part from the rest.

The next example shows a backward compatible version.

```
Example 3:

--> filename: Kbuild
obj m := 8123.o
8123-y := 8123_if.o 8123_pci.o 8123_bin.o

--> filename: Makefile
ifneq ($(KERNELRELEASE),)
# kbuild part of makefile
include Kbuild

else

# normal makefile

KDIR ?= /lib/modules/‘uname r‘/build
default:
$(MAKE) C $(KDIR) M=$$PWD
# Module specific targets
genbin:
echo "X" > 8123_bin.o_shipped
endif
```

Here the "Kbuild" file is included from the makefile. This allows an older version of kbuild, which only nows of makefiles, to be used when the "make" and kbuild parts are split into separate files.

Binary Blobs

Some external modules need to include an object file as a blob. kbuild has support for this, but requires the blob file to be named <filename>_shipped. When the kbuild rules kick in, a copy of <filename>_shipped is created with _shipped stripped off, giving us <filename>. This shortened filename can be used in the assignment to the module.

Throughout this section, 8123_bin.o_shipped has been used to build the kernel module 8123.ko; it has been included as 8123_bin.o.

```
8123 y := 8123_if.o 8123_pci.o 8123_bin.o
```

Although there is no distinction between the ordinary source files and the binary file, kbuild will pick up different rules when creating the object file for the module.

Building Multiple Modules

kbuild supports building multiple modules with a single build file. For example, if you wanted to build two modules, foo.ko and bar.ko, the kbuild lines would be:

```
obj m := foo.o bar.o
295 foo y := <foo_srcs>
bar y := <bar_srcs>
It is that simple!
```

Include Files

Within the kernel, header files are kept in standard locations according to the following rule:

- If the header file only describes the internal interface of a module, then the file is placed in the same directory as the source files.

- If the header file describes an interface used by other parts 310 of the kernel that are located in different directories, then the file is placed in include/linux/.

NOTE: There are two notable exceptions to this rule: larger subsystems have their own directory under include/, such as 315 include/scsi; and architecture specific headers are located under arch/$(ARCH)/include/.

Kernel Includes

To include a header file located under include/linux/, simply use:

```
#include <linux/module.h>
```

kbuild will add options to "gcc" so the relevant directories are searched.

Single Subdirectory

External modules tend to place header files in a separate include/ directory where their source is located, although this is not the usual kernel style. To inform kbuild of the directory, use either
ccflags-y or CFLAGS_<filename>.o.

Using the example from section 3, if we moved 8123_if.h to a subdirectory named include, the resulting kbuild file would look like:

```
--> filename: Kbuild
obj m := 8123.o
ccflags y := Iinclude
8123 y := 8123_if.o 8123_pci.o 8123_bin.o
```

Note that in the assignment there is no space between I and the path. This is a limitation of kbuild: there must be no space present.

Several Subdirectories

kbuild can handle files that are spread over several directories. Consider the following example:

```
|__ src
| |__ complex_main.c
| |__ hal
| |__ hardwareif.c
| |__ include
| |__ hardwareif.h
|__ include
|__ complex.h
```

To build the module complex.ko, we then need the following kbuild file:

```
--> filename: Kbuild
obj m := complex.o
complex y := src/complex_main.o
complex y += src/hal/hardwareif.o
ccflags y := I$(src)/include
ccflags y += I$(src)/src/hal/include
```

As you can see, kbuild knows how to handle object files located in other directories. The trick is to specify the directory relative to the kbuild file's location. That being said, this is NOT recommended practice.

For the header files, kbuild must be explicitly told where to look. When kbuild executes, the current directory is always the root of the kernel tree (the argument to " C") and therefore an absolute path is needed. $(src) provides the absolute path by pointing to the directory where the currently executing kbuild file is located.

Module Installation

Modules which are included in the kernel are installed in the directory:

```
/lib/modules/$(KERNELRELEASE)/kernel/
```

And external modules are installed in:

```
/lib/modules/$(KERNELRELEASE)/extra/
```

INSTALL_MOD_PATH

Above are the default directories but as always some level of customization is possible. A prefix can be added to the installation path using the variable INSTALL_MOD_PATH:

```
$ make INSTALL_MOD_PATH=/frodo modules_install
=> Install dir: /frodo/lib/modules/$(KERNELRELEASE)/kernel/
```

INSTALL_MOD_PATH may be set as an ordinary shell variable or, as shown above, can be specified on the command line when calling "make." This has effect when installing both in tree and out of tree modules.

INSTALL_MOD_DIR

External modules are by default installed to a directory under /lib/modules/$(KERNELRELEASE)/extra/, but you may wish to locate modules for a specific functionality in a separate directory. For this purpose, use INSTALL_MOD_DIR to specify an alternative name to "extra."

```
$ make INSTALL_MOD_DIR=gandalf C $KDIR \
M=$PWD modules_install
=> Install dir: /lib/modules/$(KERNELRELEASE)/gandalf/
```

Module Versioning

Module versioning is enabled by the CONFIG_MODVERSIONS tag, and is used as a simple ABI consistency check. A CRC value of the full prototype for an exported symbol is created. When a module is loaded/used, the CRC values contained in the kernel are compared with similar values in the module; if they are not equal, the kernel refuses to load the module.

Module.symvers contains a list of all exported symbols from a kernel build.

Symbols From the Kernel (vmlinux + modules)

During a kernel build, a file named Module.symvers will be generated. Module.symvers contains all exported symbols from the kernel and compiled modules. For each symbol, the corresponding CRC value is also stored.

The syntax of the Module.symvers file is:

```
<CRC> <Symbol> <module>
0x2d036834 scsi_remove_host drivers/scsi/scsi_mod
```

For a kernel build without CONFIG_MODVERSIONS enabled, the CRC would read 0x00000000.
Module.symvers serves two purposes:

1) It lists all exported symbols from vmlinux and all modules.
2) It lists the CRC if CONFIG_MODVERSIONS is enabled.

Symbols and External Modules

When building an external module, the build system needs access to the symbols from the kernel to check if all external symbols are defined. This is done in the MODPOST step. modpost obtains the symbols by reading Module.symvers from the kernel source tree. If a Module.symvers file is present in the directory where the external module is being built, this file will be 465 read too. During the MODPOST step, a new Module.symvers file will be written containing all exported symbols that were not defined in the kernel.

Symbols From Another External Module

Sometimes, an external module uses exported symbols from another external module. kbuild needs to have full knowledge of all symbols to avoid spitting out warnings about undefined
symbols. Three solutions exist for this situation.

NOTE: The method with a top level kbuild file is recommended but may be impractical in certain situations.

Use a top level kbuild file

If you have two modules, foo.ko and bar.ko, where foo.ko needs symbols from bar.ko, you can use a common top level kbuild file so both modules are compiled in the same build. Consider the following directory layout:

```
./foo/ <= contains foo.ko
./bar/ <= contains bar.ko
```

The top level kbuild file would then look like:

```
#./Kbuild (or ./Makefile):
obj y := foo/ bar/
```

And executing

```
$ make C $KDIR M=$PWD
```

will then do the expected and compile both modules with full knowledge of symbols from either module.

Use an extra Module.symvers file

When an external module is built, a Module.symvers file is generated containing all exported symbols which are not defined in the kernel. To get access to symbols from bar.ko, copy the Module.symvers file from the compilation of bar.ko to the directory where foo.ko is built. During the module build, kbuild will read the Module.symvers file in the directory of the external module, and when the build is finished, a new
Module.symvers file is created containing the sum of all symbols defined and not part of the kernel.

Use "make" variable KBUILD_EXTRA_SYMBOLS

If it is impractical to copy Module.symvers from 515 another module, you can assign a space separated list
of files to KBUILD_EXTRA_SYMBOLS in your build file. These files will be loaded by modpost during the
initialization of its symbol tables.

Tips & Tricks

Testing for CONFIG_FOO_BAR

Modules often need to check for certain CONFIG_ options to decide if a specific feature is included in the module. In kbuild this is done by referencing the CONFIG_ variable directly.

```
#fs/ext2/Makefile
obj $(CONFIG_EXT2_FS) += ext2.o

ext2 y := balloc.o bitmap.o dir.o
ext2 $(CONFIG_EXT2_FS_XATTR) += xattr.o
```

External modules have traditionally used "grep" to check for specific CONFIG_ settings directly in .config. This usage is broken. As introduced before, external modules should use kbuild for building and can therefore use the same methods as in tree modules when testing for CONFIG_ definitions.

5.2.2.2.1.2) Cross-Compiling LiME for Android

In order to cross-compile LiME for use on an Android device, few additional steps are required:

- Install the general android prerequisites.
- Download and un (zip|tar) the android NDK.
- Download and un (zip|tar) the android SDK.
- Download and un-tar the kernel source for your device. This can usually be found on the website of your device manufacturer or by a quick Google search.
- Root the device. In order to run custom kernel modules, we must have a rooted device.
- Plug the device into computer via a USB cable.

Setting up the Environment

- In order to simplify the process, we will first set some environment variables. In a terminal, type the following commands:

```
▪ $ export SDK_PATH=/path/to/android-sdk-linux/
▪ $ export NDK_PATH=/path/to/android-ndk/ $ export KSRC_PATH=/path/to/kernel-source/
▪ $ export CC_PATH=$NDK_PATH/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86/bin/
▪ $ export LIME_SRC=/path/to/lime/src
```

Preparing the Kernel Source

- We must retrieve and copy the kernel config from our device:

```
$ cd $SDK_PATH/platform-tools
$ ./adb pull /proc/config.gz
$ gunzip ./config.gz
$ cp config $KSRC_PATH/.config
```

- Next, we have to prepare our kernel source for our module:

```
$ cd $KSRC_PATH
$ make ARCH=arm CROSS_COMPILE=$CC_PATH/arm-eabi- modules_prepare
```

Preparing the Module Compilation

- We need to create a Makefile to cross-compile our kernel module. A sample Makefile for cross-compiling is shipped with the LiME source. The contents of your Makefile should be similar to the following:

```
obj-m := lime.o
lime-objs := main.o tcp.o disk.o
KDIR := /path/to/kernel-source

PWD := $(shell pwd)

CCPATH := /path/to/android-ndk/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86/bin/

default:

	$(MAKE) ARCH=arm CROSS_COMPILE=$(CCPATH)/arm-eabi- -C $(KDIR) M=$(PWD) modules
```

Compiling the Module

```
$ cd $LIME_SRC

$ make
```

5.2.2.2.2) Using (or Running) LiME

To illustrate the use of LiME, we will now walk through two examples of acquiring memory from an Android device. First we will discuss the acquisition of memory over a TCP connection, followed by a discussion of acquiring a memory dump via the device's SD card. The use of LiME on other Linux devices is similar, however, the use of the Android debug bridge (*adb*) is not needed.

Starting in version 1.1, LiME supports multiple output formats, including a custom lime format which integrates with Volatility's new lime address space. This means that additional parameters are needed when installing the LiME kernel module.

NOTE: There is a bug in the ***insmod*** utility on some Android devices. Multiple kernel module parameters must be wrapped in quotation marks, otherwise only the first parameter will be parsed.

LiME Parameters

• path
    – Either a filename to write on the local system (SD Card) or tcp:<port>
• format

    – raw
• Simply concatenates all System RAM ranges

    – padded
• Pads all non-System RAM ranges with 0s, starting from physical address 0

    – lime
• Each range is prepended with a fixed-sized header which contains address space information
• Volatility address space developed to support this format
• dio (optional)

    – 1 to enable Direct IO attempt (default), 0 to disable

5.2.2.2.2.1) Acquisition of Memory over TCP

The first step of the process is to copy the kernel module to the device's SD card using the Android Debug Bridge (*adb*), which supports a number of interactions with an Android device tethered via USB. We then use *adb* to setup a port-forwarding tunnel from a TCP port on the device to a TCP port on the local host. The use of *adb* for network transfer eliminates the need to modify the networking configuration on the device or introduce a wireless peer—all network data is transferred via USB. For the example below, we have chosen TCP port **4444**. We then obtain a root shell on the device by using *adb* and *su*. To accomplish this, we run the following commands with the phone plugged into our computer and debugging enabled on the device.

```
$ adb push lime.ko /sdcard/lime.ko
$ adb forward tcp:4444 tcp:4444
$ adb shell
$ su
#
```

Memory acquisition over the TCP tunnel is then a two-part process. First, the target device must listen on a specified TCP port and then we must connect to the device from the host computer. When the socket is connected, the kernel module will automatically send the acquired RAM image to the host device.

In the *adb* root shell, we install our kernel module using the *insmod* command. To instruct the module to dump memory via TCP, we set the path parameter to "tcp", followed by a colon and then the port number that *adb* is forwarding. On our host computer, we connect to this port with *netcat* and redirect output to a file. We also select the "lime" formatting option. When the acquisition process is complete, LiME will terminate the TCP connection.

The following command loads the kernel module via *adb* on the target Android device:

```
# insmod  /sdcard/lime.ko  "path=tcp:4444 format=lime"
```

On the host, the following command captures the memory dump via TCP port 444 to the file "ram.lime":

```
$  nc localhost 4444  >  ram.lime
```

5.2.2.2.2.2) Acquisition of Memory to Disk (SD Card)

In some cases, such as when the investigator wants to make sure no network buffers are overwritten, disk-based acquisition may be preferred to network acquisition. To accommodate this situation, LiME provides the option to write memory images to the device's file system. On Android, the logical place to write is the device's SD card.

Since the SD card could potentially contain other relevant evidence to the case, the investigator may wish to take an image from the SD card first in order to save unallocated space. Unfortunately, some Android phones, such as the HTC EVO 4G and the Droid series, place the removable SD card to be either under or obstructed by the phone's battery, making it impossible to remove the SD card without powering off the phone (these phones will power down if the battery is removed, even if they are plugged into a power source!). For this reason, the investigator needs to first image the SD card, and then subsequently write the memory image to it. While this process violates the typical "order of volatility" rule of thumb in forensic acquisition, namely, obtaining the most volatile information first, it is necessary to properly preserve all evidence.

Fortunately, take an image from the SD card on an Android device that will be subjected to live forensic analysis (including memory dumping) does not require removal of the SD card. Tethering the device to a Linux machine, for example, and activating USB Storage exposes a /dev/sd device that can be imaged using traditional means (e.g., using dd on the Linux box). Activating USB Storage mode unmounts the SD card on the Android device, so a forensically valid image can be obtained.

With USB Storage mode deactivated, we copy the LiME kernel module to the device using the same steps described in the last section. When installing the module using ***insmod***, we set the path parameter to /sdcard/ram.lime to specify the file in which to write the memory dump. We also select the "lime" format option:

```
# insmod /sdcard/lime.ko "path=/sdcard/ram.lime format=lime"
```

Once the acquisition process is complete, we can power down the phone, remove the SD card from the phone, and transfer the memory dump to the examination machine. If the phone cannot be powered down, *adb* can also be used to transfer the memory dump to the investigator's machine.

5.2.2.2.3) LiME forensics acquisition

Before moving on to the technical part of the LiME forensic tool, we must clear that we created a common test account for various applications (e.g. Facebook, twitter, gmail, youtube) and we log-in to each account so that credentials be easily identifiable. However we should activate the USB Debugging. So we should navigate on mobile device to the following path: **settings→ Developer options**:

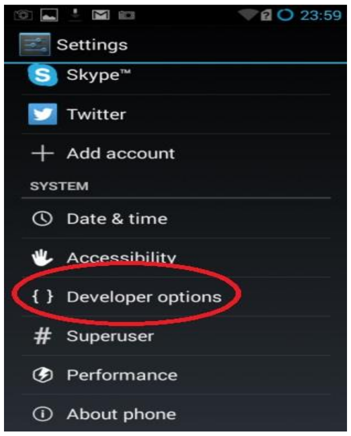
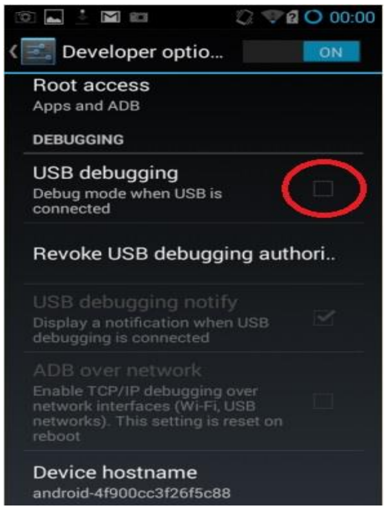


Figure 126

- After that, we should activate the USB Debugging option, which :



Root access
Apps and ADB

DEBUGGING

USB debugging
Debug mode when USB is connected

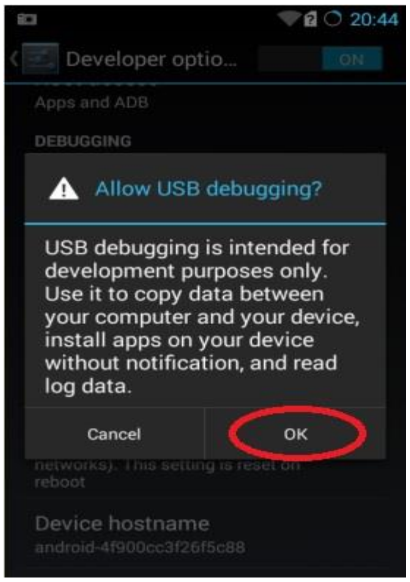
Revoke USB debugging authori..

USB debugging notify
Display a notification when USB debugging is connected

ADB over network
Enable TCP/IP debugging over network interfaces (Wi-Fi, USB networks). This setting is reset on reboot

Device hostname
android-4f900cc3f26f5c88

Figure 127



20:44

Developer optio... ON

Apps and ADB

DEBUGGING

Allow USB debugging?

USB debugging is intended for development purposes only. Use it to copy data between your computer and your device, install apps on your device without notification, and read log data.

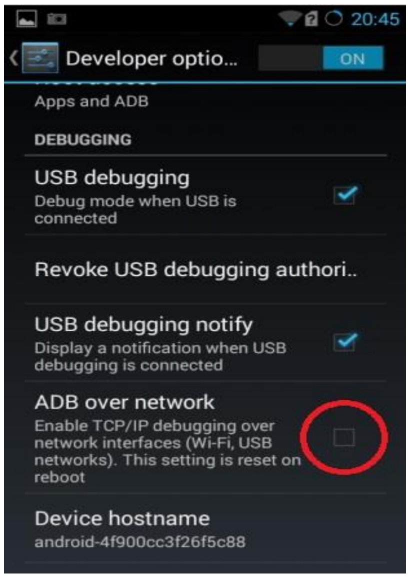
Cancel OK

networks). This setting is reset on reboot

Device hostname
android-4f900cc3f26f5c88

Figure 128

- Also we must activate the ADB over network, which:



20:45

Developer optio... ON

Apps and ADB

DEBUGGING

USB debugging ✓
Debug mode when USB is connected

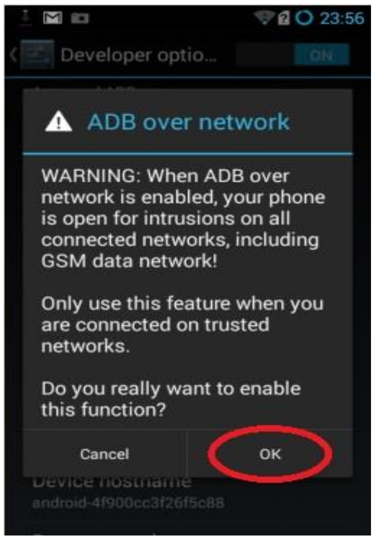
Revoke USB debugging authori..

USB debugging notify ✓
Display a notification when USB debugging is connected

ADB over network ☐
Enable TCP/IP debugging over network interfaces (Wi-Fi, USB networks). This setting is reset on reboot

Device hostname
android-4f900cc3f26f5c88

Figure 129



23:56

Developer optio... ON

ADB over network

WARNING: When ADB over network is enabled, your phone is open for intrusions on all connected networks, including GSM data network!

Only use this feature when you are connected on trusted networks.

Do you really want to enable this function?

Figure 130

- For our technical track, we will need the function of ADB, which we downloaded it (via the internet) and placed it in a particular path (home/android-sdk-linux/platform-tools) in order to perform it later:

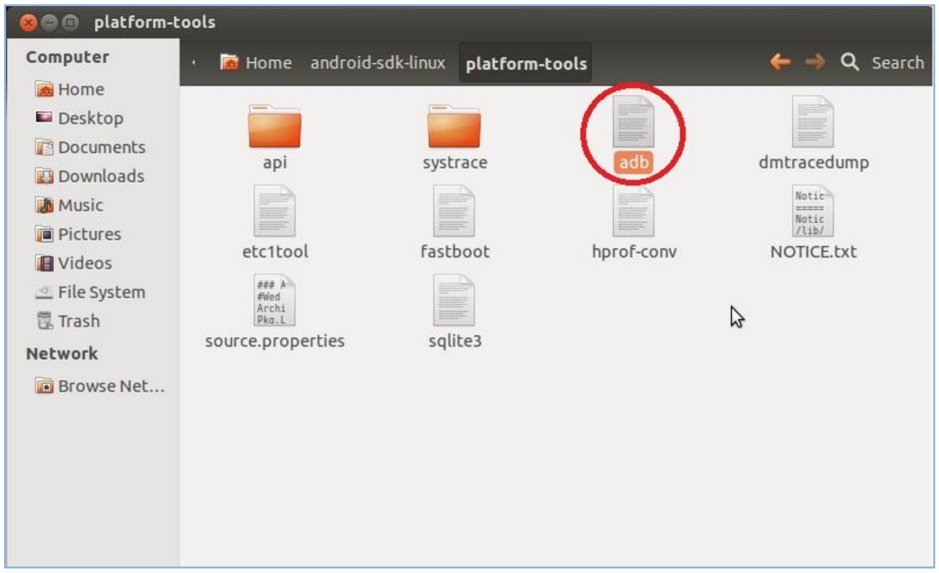
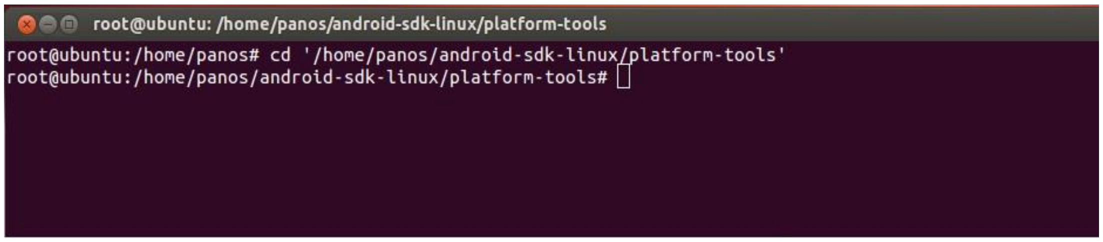


Figure 131

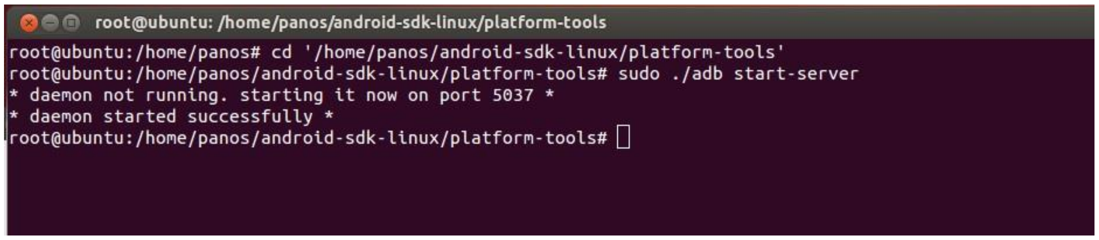
- We are going to the appropriate path (in which the ADB is located):



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
root@ubuntu:/home/panos# cd '/home/panos/android-sdk-linux/platform-tools'
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# 
```

Figure 132

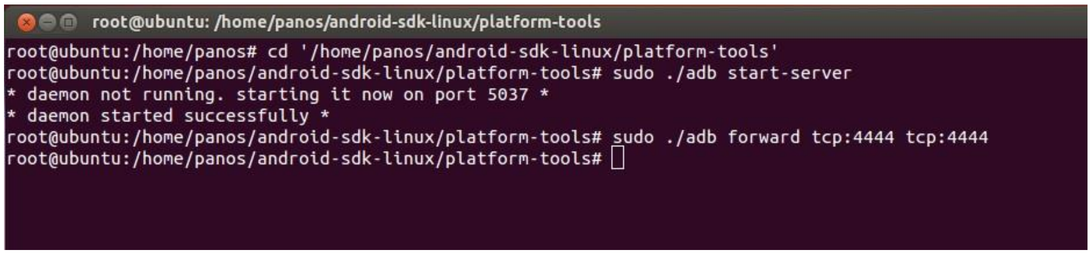
- We start the ADB server:



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
root@ubuntu:/home/panos# cd '/home/panos/android-sdk-linux/platform-tools'
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# 
```

Figure 133

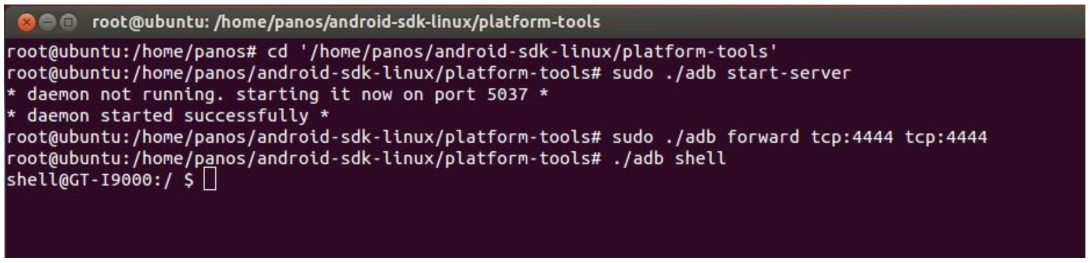
- We create a listener (sudo ./adb forward tcp:4444 tcp:4444) between device and computer:



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
root@ubuntu:/home/panos# cd '/home/panos/android-sdk-linux/platform-tools'
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb forward tcp:4444 tcp:4444
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# 
```

Figure 134

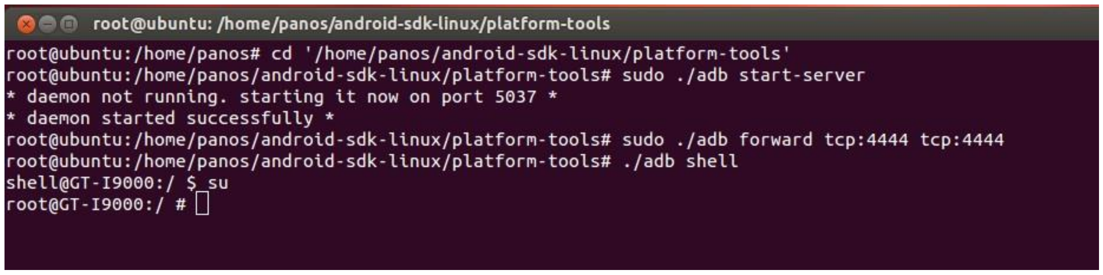
- We open a shell (./adb shell) for having a direct connection with the device:



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
root@ubuntu:/home/panos# cd '/home/panos/android-sdk-linux/platform-tools'
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb forward tcp:4444 tcp:4444
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# ./adb shell
shell@GT-I9000:/ $ 
```

Figure 135

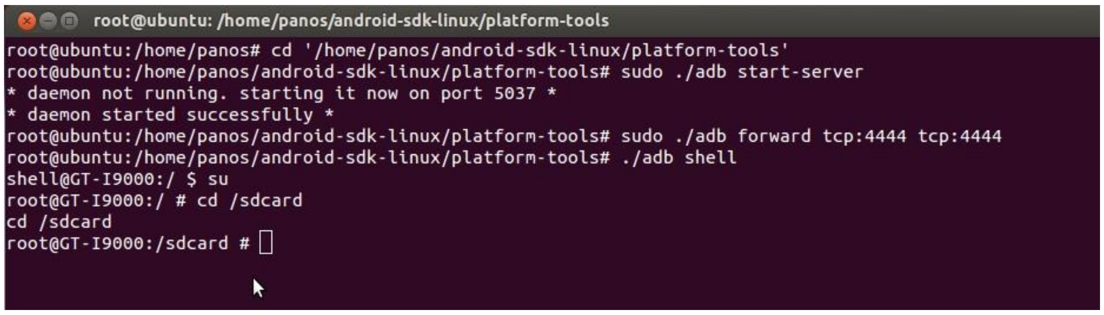
- We need super user permissions (su):



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
root@ubuntu:/home/panos# cd '/home/panos/android-sdk-linux/platform-tools'
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb forward tcp:4444 tcp:4444
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# ./adb shell
shell@GT-I9000:/ $ su
root@GT-I9000:/ # 
```

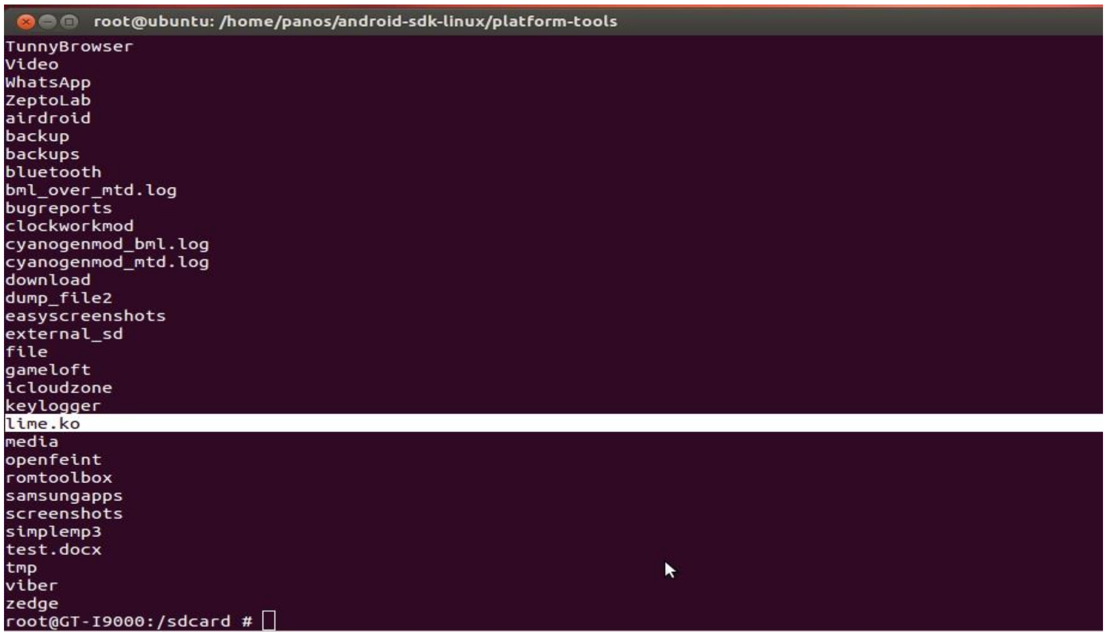
Figure 136

- We should navigate (cd /sdcard) in the correct path in order to fine the LiME tool:



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
root@ubuntu:/home/panos# cd '/home/panos/android-sdk-linux/platform-tools'
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# sudo ./adb forward tcp:4444 tcp:4444
root@ubuntu:/home/panos/android-sdk-linux/platform-tools# ./adb shell
shell@GT-I9000:/ $ su
root@GT-I9000:/ # cd /sdcard
cd /sdcard
root@GT-I9000:/sdcard # 
```

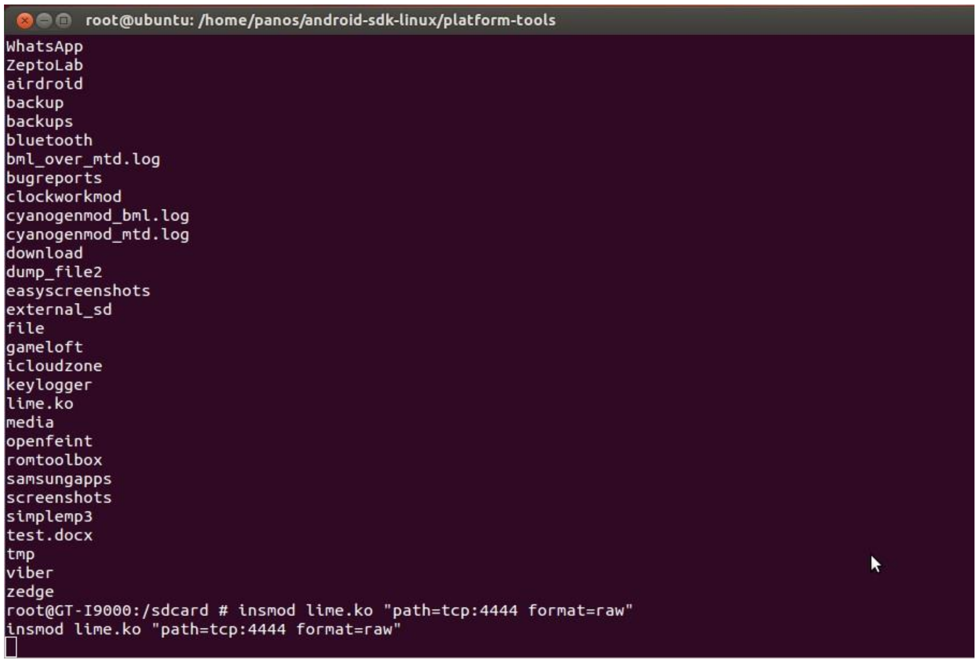
Figure 137



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
TunnyBrowser
Video
WhatsApp
ZeptoLab
airdroid
backup
backups
bluetooth
bml_over_mtd.log
bugreports
clockworkmod
cyanogenmod_bml.log
cyanogenmod_mtd.log
download
dump_file2
easyscreenshots
external_sd
file
gameloft
icloudzone
keylogger
lime.ko
media
openfeint
romtoolbox
samsungapps
screenshots
simplemp3
test.docx
tmp
viber
zedge
root@GT-I9000:/sdcard #
```

Figure 138

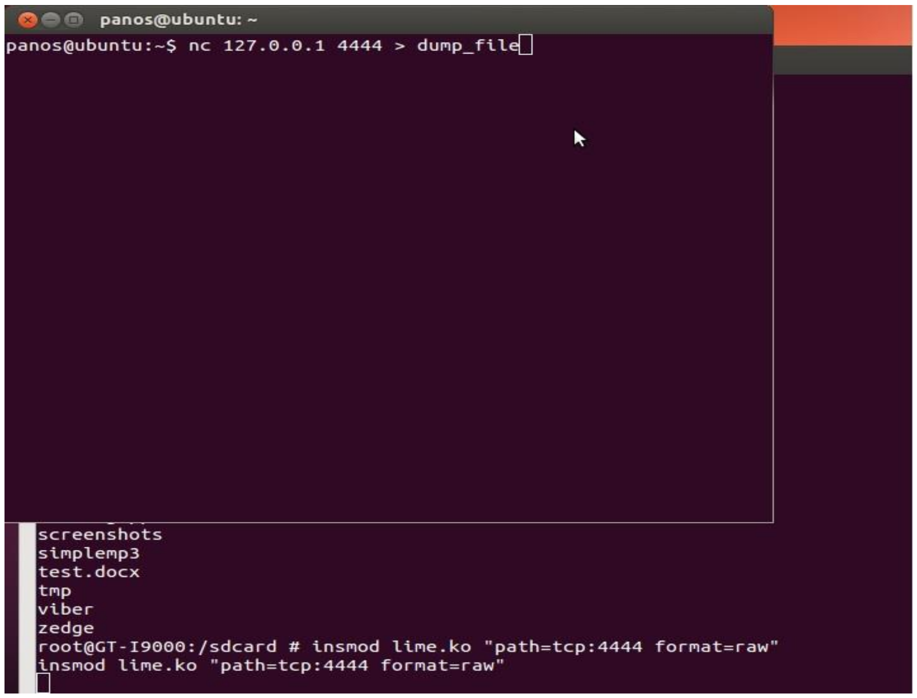
- (insmod lime.ko “path=tcp:4444 format=raw”):



```
root@ubuntu: /home/panos/android-sdk-linux/platform-tools
WhatsApp
ZeptoLab
airdroid
backup
backups
bluetooth
bml_over_mtd.log
bugreports
clockworkmod
cyanogenmod_bml.log
cyanogenmod_mtd.log
download
dump_file2
easyscreenshots
external_sd
file
gameloft
icloudzone
keylogger
lime.ko
media
openfeint
romtoolbox
samsungapps
screenshots
simplemp3
test.docx
tmp
viber
zedge
root@GT-I9000:/sdcard # insmod lime.ko "path=tcp:4444 format=raw"
insmod lime.ko "path=tcp:4444 format=raw"
```

Figure 139

- In a second terminal we extract (nc 127.0.0.1 4444 > dump_file) the whole memory dump from the smartphone to a file (dump_file):

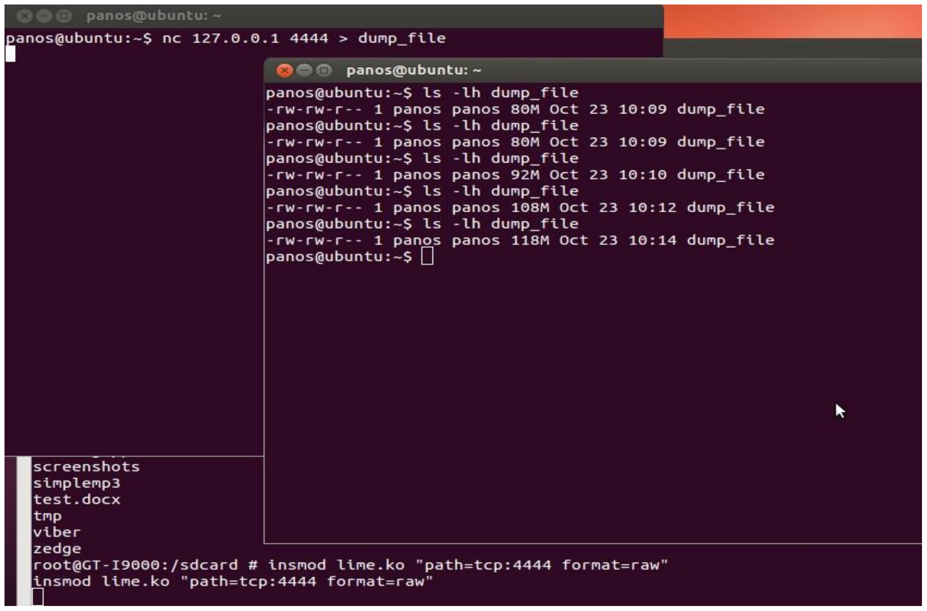


```
panos@ubuntu: ~
panos@ubuntu:~$ nc 127.0.0.1 4444 > dump_file
```

```
screenshots
simplemp3
test.docx
tmp
viber
zedge
root@GT-I9000:/sdcard # insmod lime.ko "path=tcp:4444 format=raw"
insmod lime.ko "path=tcp:4444 format=raw"
```

Figure 140

- In third one, we check if the file size (dump_file) normally increases (ls –lh dump_file):



```
panos@ubuntu: ~
panos@ubuntu:~$ nc 127.0.0.1 4444 > dump_file
```

```
panos@ubuntu: ~
panos@ubuntu:~$ ls -lh dump_file
-rw-rw-r-- 1 panos panos 80M Oct 23 10:09 dump_file
panos@ubuntu:~$ ls -lh dump_file
-rw-rw-r-- 1 panos panos 80M Oct 23 10:09 dump_file
panos@ubuntu:~$ ls -lh dump_file
-rw-rw-r-- 1 panos panos 92M Oct 23 10:10 dump_file
panos@ubuntu:~$ ls -lh dump_file
-rw-rw-r-- 1 panos panos 108M Oct 23 10:12 dump_file
panos@ubuntu:~$ ls -lh dump_file
-rw-rw-r-- 1 panos panos 118M Oct 23 10:14 dump_file
panos@ubuntu:~$
```

```
screenshots
simplemp3
test.docx
tmp
viber
zedge
root@GT-I9000:/sdcard # insmod lime.ko "path=tcp:4444 format=raw"
insmod lime.ko "path=tcp:4444 format=raw"
```

Figure 141

- When the process is completed then we check if the file was created successfully:

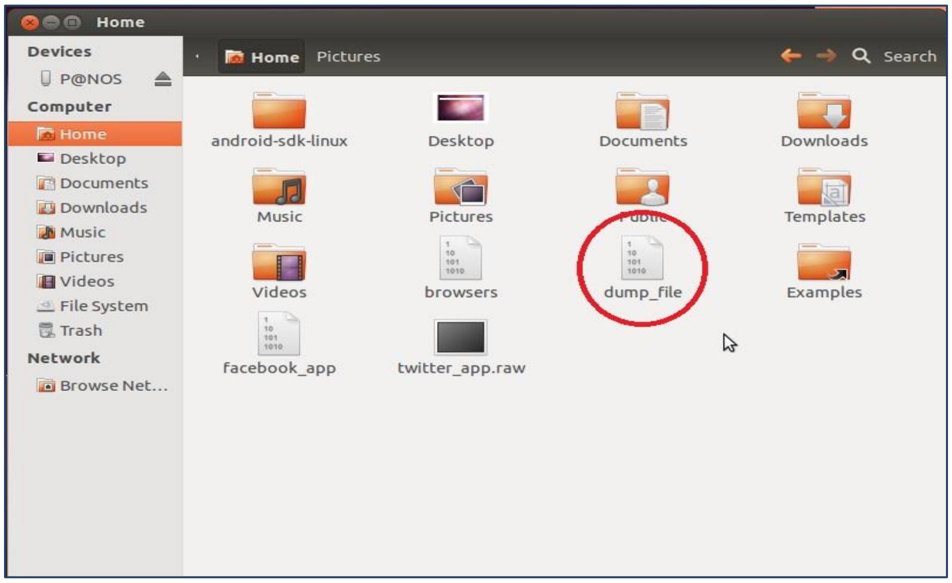
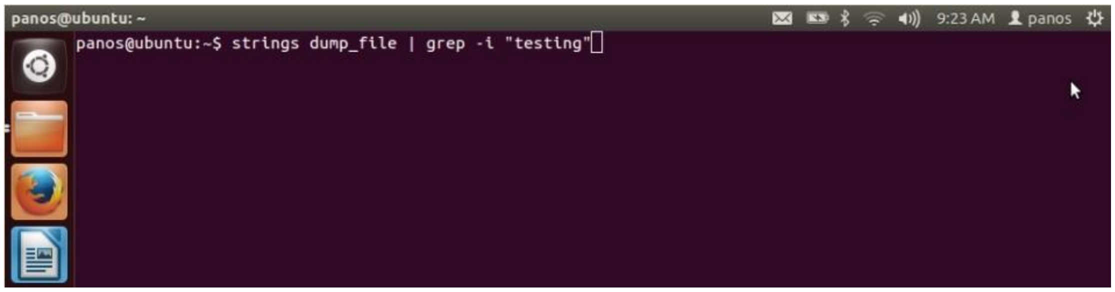


Figure 142

Once having the export process memory ROM successfully completed then we proceed to scrutinize the file (containing the memory of the cell) for finding possible user's credentials from various applications with word keys.

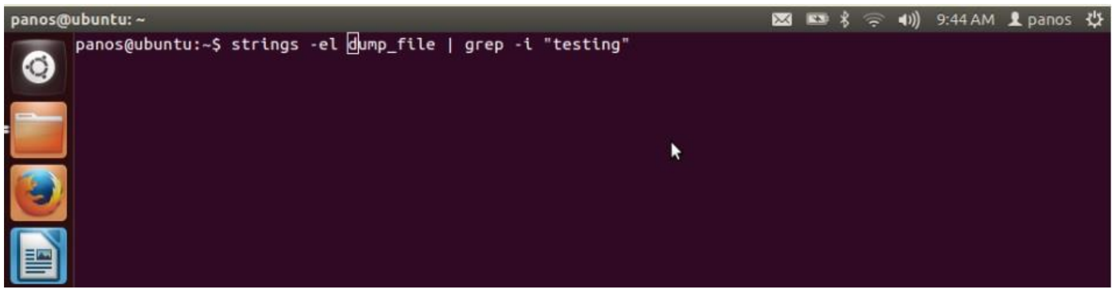
- We check the file with key words (strings dump_file | grep –I “testing”):



```
panos@ubuntu:~$ strings dump_file | grep -i "testing"
```

Figure 143

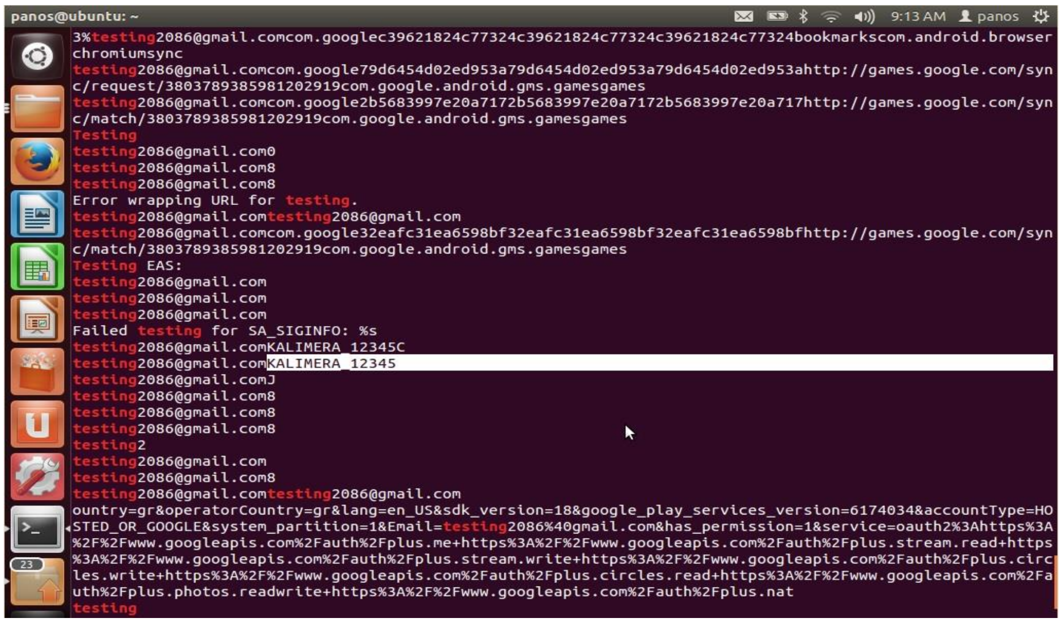
- We check for Unicode characters (strings –el dump_file | grep –I “testing”) in the file:



```
panos@ubuntu:~$ strings -el dump_file | grep -i "testing"
```

Figure 144

- After executing the above command (strings –el dump_file | grep –I “testing”), the results are the following:

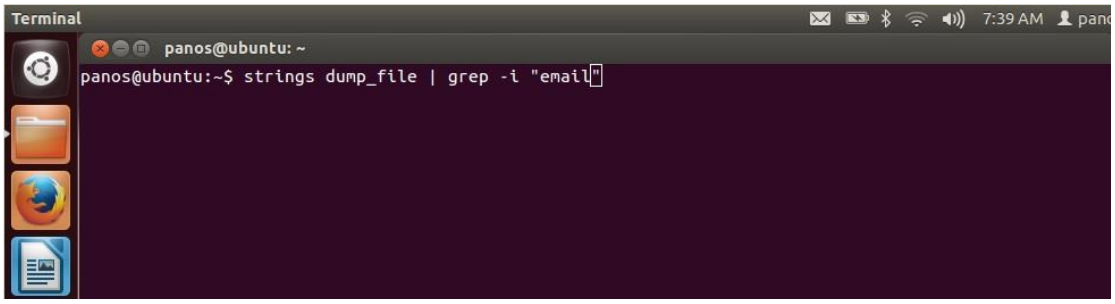


```
3%testing2086@gmail.comcom.googlec39621824c77324c39621824c77324c39621824c77324bookmarkscom.android.browser
chromiumsync
testing2086@gmail.comcom.google79d6454d02ed953a79d6454d02ed953a79d6454d02ed953ahttp://games.google.com/syn
c/request/3803789385981202919com.google.android.gms.gamesgames
testing2086@gmail.comcom.google2b5683997e20a7172b5683997e20a7172b5683997e20a717http://games.google.com/syn
c/match/3803789385981202919com.google.android.gms.gamesgames
Testing
testing2086@gmail.com0
testing2086@gmail.com8
testing2086@gmail.com8
Error wrapping URL for testing.
testing2086@gmail.comtesting2086@gmail.com
testing2086@gmail.comcom.google32eafc31ea6598bf32eafc31ea6598bf32eafc31ea6598bfhttp://games.google.com/syn
c/match/3803789385981202919com.google.android.gms.gamesgames
Testing EAS:
testing2086@gmail.com
testing2086@gmail.com
testing2086@gmail.com
Failed testing for SA_SIGINFO: %s
testing2086@gmail.comKALIMERA_12345C
testing2086@gmail.comKALIMERA_12345
testing2086@gmail.comJ
testing2086@gmail.com8
testing2086@gmail.com8
testing2086@gmail.com8
testing2
testing2086@gmail.com
testing2086@gmail.com8
testing2086@gmail.comtesting2086@gmail.com
ountry=gr&operatorCountry=gr&lang=en_US&sdk_version=18&google_play_services_version=6174034&accountType=HO
STED_OR_GOOGLE&system_partition=1&Email=testing2086%40gmail.com&has_permission=1&service=oauth2%3Ahttps%3A
%2F%2Fwww.googleapis.com%2Fauth%2Fplus.me+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fplus.stream.read+https
%3A%2F%2Fwww.googleapis.com%2Fauth%2Fplus.stream.write+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fplus.circ
les.write+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fplus.circles.read+https%3A%2F%2Fwww.googleapis.com%2Fa
uth%2Fplus.photos.readwrite+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fplus.nat
testing
```

Figure 145

According to the above results we managed to distract the email address (testing2086@gmail.com) and the password of the user (**KALIMERA_12345**).

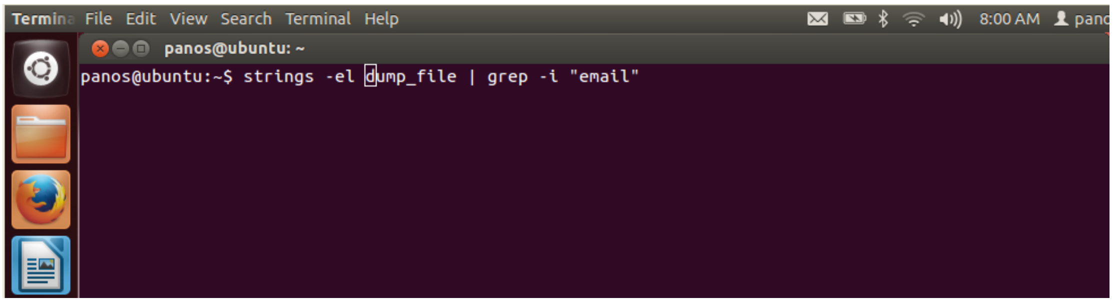
- We check with different key words (strings dump_file | grep –I “email”):



```
panos@ubuntu:~$ strings dump_file | grep -i "email"
```

Figure 146

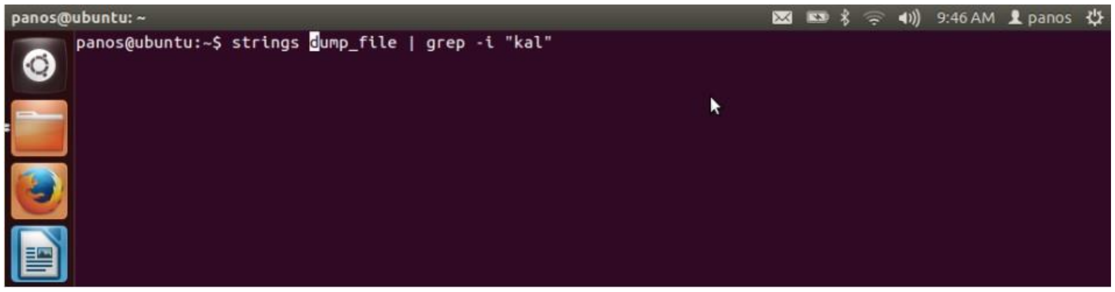
- And for unicodes characters (strings –el dump_file | grep –I “email”):



```
panos@ubuntu:~$ strings -el dump_file | grep -i "email"
```

Figure 147

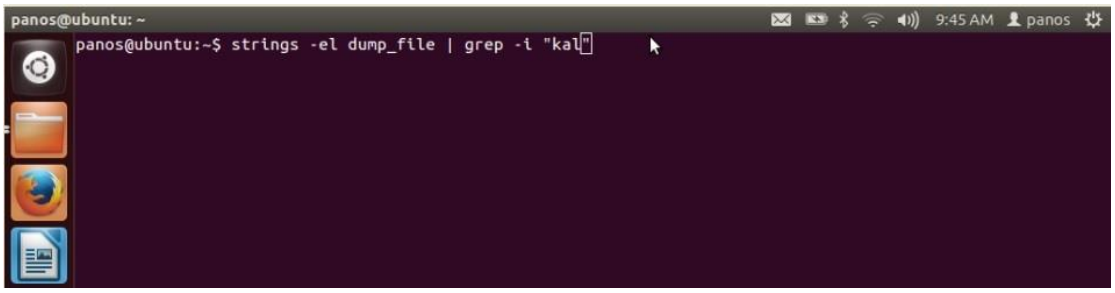
- And with another key word (strings dump_file | grep –I “kal”):



```
panos@ubuntu:~$ strings dump_file | grep -i "kal"
```

Figure 148

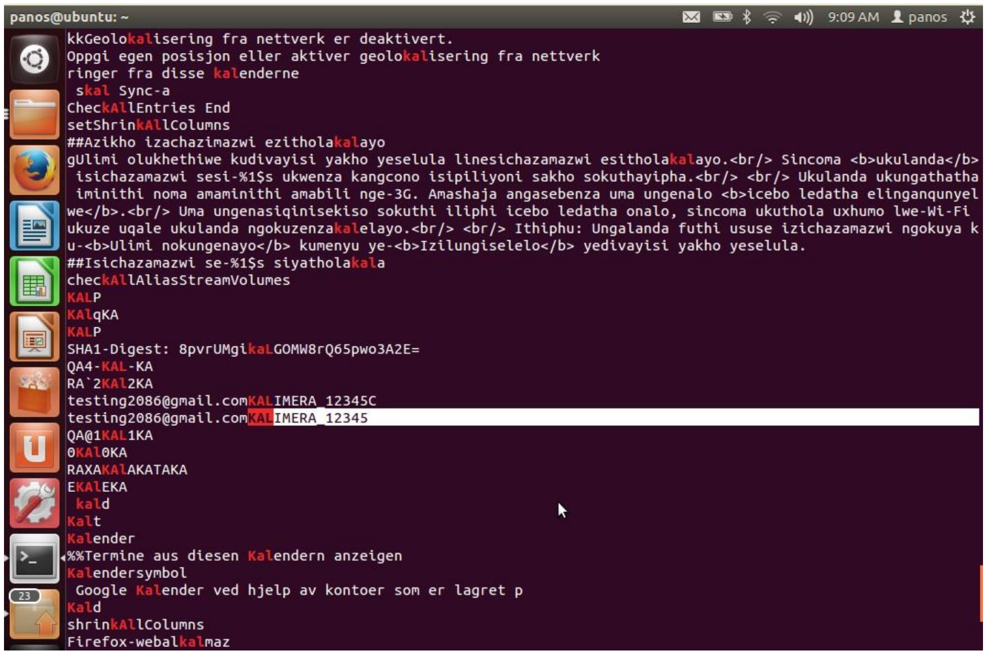
- (strings –el dump_file | grep –I “kal”):



```
panos@ubuntu:~$ strings -el dump_file | grep -i "kal"
```

Figure 149

- After executing the above command (strings –el dump_file | grep –I “kal”), the results are the following:



```
kkGeolokalisering fra nettverk er deaktivert.
Oppgi egen posisjon eller aktiver geolokalisering fra nettverk
ringer fra disse kalenderne
 skal Sync-a
CheckAllEntries End
setShrinkAllColumns
##Azikho izachazimazwi ezitholakalayo
gUlimi olukhethiwe kudivayisi yakho yeselula linesichazamazwi esitholakalayo.<br/> Sincoma <b>ukulanda</b>
 isichazamazwi sesi-%1$s ukwenza kangcono isipiliyoni sakho sokuthayipha.<br/> <br/> Ukulanda ukungathatha
 iminithi noma amaminithi amabili nge-3G. Amashaja angasebenza uma ungenalo <b>icebo ledatha elinganqunyel
we</b>.<br/> Uma ungenasiqinisekiso sokuthi iliphi icebo ledatha onalo, sincoma ukuthola uxhumo lwe-Wi-Fi
ukuze uqale ukulanda ngokuzenzakalelayo.<br/> <br/> Ithiphu: Ungalanda futhi ususe izichazamazwi ngokuya k
u-<b>Ulimi nokungenayo</b> kumenyu ye-<b>Izilungiselelo</b> yedivayisi yakho yeselula.
##Isichazamazwi se-%1$s siyatholakala
checkAllAliasStreamVolumes
KALP
KAlqKA
KALP
SHA1-Digest: 8pvrUMgikaLGOMW8rQ65pwo3A2E=
QA4-KAL-KA
RA`2KAl2KA
testing2086@gmail.comKALIMERA_12345C
testing2086@gmail.comKALIMERA_12345
QA@1KAL1KA
0KAl0KA
RAXAKAlAKATAKA
EKAlEKA
 kald
Kalt
Kalender
%%Termine aus diesen Kalendern anzeigen
Kalendersymbol
 Google Kalender ved hjelp av kontoer som er lagret p
Kald
shrinkAllColumns
Firefox-webalkalmaz
```

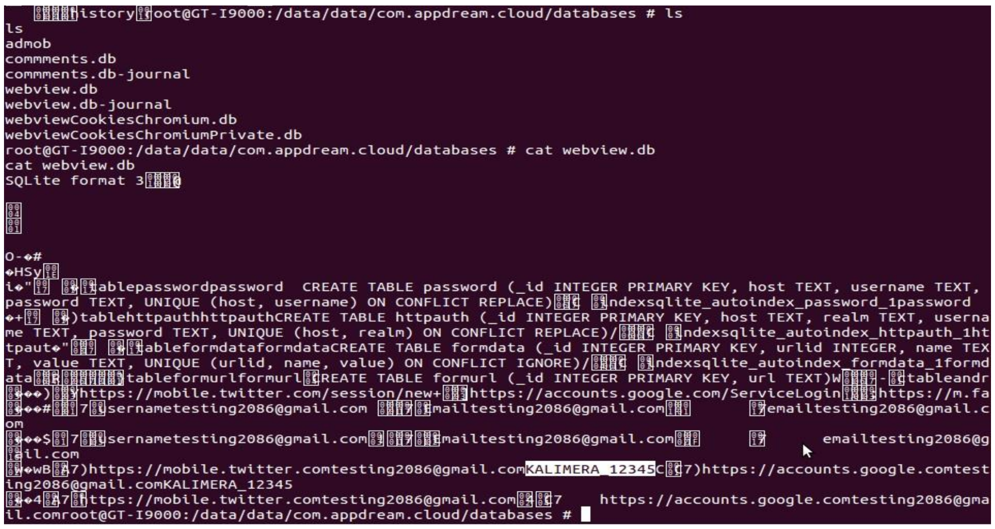
Figure 150

Again from the previous results we could reveal the email address (testing2086@gmail.com) and the password of the user (**KALIMERA_12345**).

An alternative way to find and retrieve the credentials (username: testing2086@gmail.com, password: **KALIMERA12345**) is the connection (via cable) directly to mobile device. After the successful connection (via shell), we navigate to the path: /data/data which contains all the applications of the device.

Since the majority of the applications were able to hide successfully the user's credentials, although some of these as cloud browser, android browser (default), twitter could not.

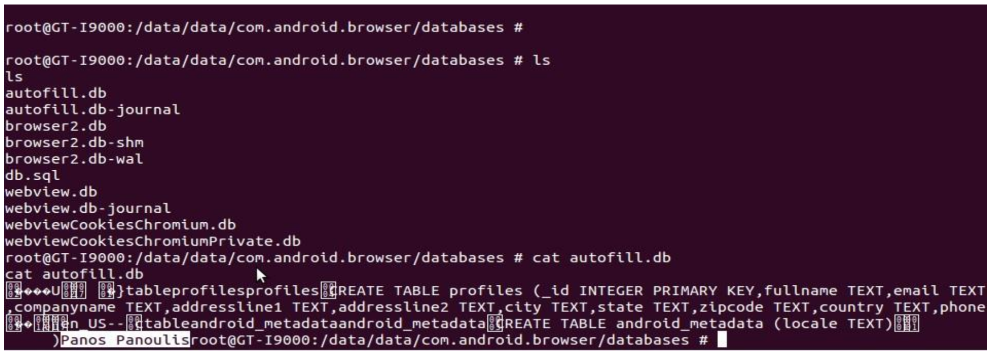
- **Cloud Browser** (data/data/com.appdream.cloud/databases/webview.db):



```
   ����history�oot@GT-I9000:/data/data/com.appdream.cloud/databases # ls
ls
admob
commments.db
commments.db-journal
webview.db
webview.db-journal
webviewCookiesChromium.db
webviewCookiesChromiumPrivate.db
root@GT-I9000:/data/data/com.appdream.cloud/databases # cat webview.db
cat webview.db
SQLite format 3����

�
�

O-�#
�HSy�
i�"� ��tablepasswordpassword  CREATE TABLE password (_id INTEGER PRIMARY KEY, host TEXT, username TEXT,
password TEXT, UNIQUE (host, username) ON CONFLICT REPLACE)�� �ndexsqlite_autoindex_password_1password
�+� ��)tablehttpauthhttpauthCREATE TABLE httpauth (_id INTEGER PRIMARY KEY, host TEXT, realm TEXT, userna
me TEXT, password TEXT, UNIQUE (host, realm) ON CONFLICT REPLACE)/���� �ndexsqlite_autoindex_httpauth_1ht
tpaut�"�� ���tableformdataformdataCREATE TABLE formdata (_id INTEGER PRIMARY KEY, urlid INTEGER, name TEX
T, value TEXT, UNIQUE (urlid, name, value) ON CONFLICT IGNORE)/���� �ndexsqlite_autoindex_formdata_1formd
ata��R�������tableformurlformurl�CREATE TABLE formurl (_id INTEGER PRIMARY KEY, url TEXT)W����-��tableandr
���)��https://mobile.twitter.com/session/new+��https://accounts.google.com/ServiceLogin���https://m.fa
���#��7�sernametesting2086@gmail.com ��7�mailtesting2086@gmail.com�      �emailtesting2086@gmail.c
om
���$�7��sernametesting2086@gmail.com�� ��7��mailtesting2086@gmail.com��     �        emailtesting2086@g
�ail.com
��wB�7)https://mobile.twitter.comtesting2086@gmail.comKALIMERA_12345C��7)https://accounts.google.comtest
ing2086@gmail.comKALIMERA_12345
��4�7�ttps://mobile.twitter.comtesting2086@gmail.com�4�7    https://accounts.google.comtesting2086@gma
il.comroot@GT-I9000:/data/data/com.appdream.cloud/databases # █
```

Figure 151

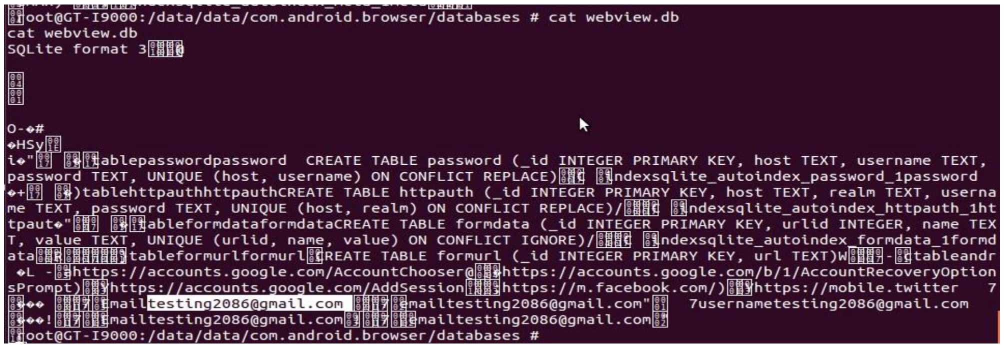
- **Default Android browser** (data/data/com.android.browser/databases/autofill.db):



```
root@GT-I9000:/data/data/com.android.browser/databases #

root@GT-I9000:/data/data/com.android.browser/databases # ls
ls
autofill.db
autofill.db-journal
browser2.db
browser2.db-shm
browser2.db-wal
db.sql
webview.db
webview.db-journal
webviewCookiesChromium.db
webviewCookiesChromiumPrivate.db
root@GT-I9000:/data/data/com.android.browser/databases # cat autofill.db
cat autofill.db
����U�� ��}tableprofilesprofiles�CREATE TABLE profiles (_id INTEGER PRIMARY KEY,fullname TEXT,email TEXT
,companyname TEXT,addressline1 TEXT,addressline2 TEXT,city TEXT,state TEXT,zipcode TEXT,country TEXT,phone
���en_US--�tableandroid_metadataandroid_metadata�CREATE TABLE android_metadata (locale TEXT)��
     )Panos Panoulisroot@GT-I9000:/data/data/com.android.browser/databases # █
```

Figure 152

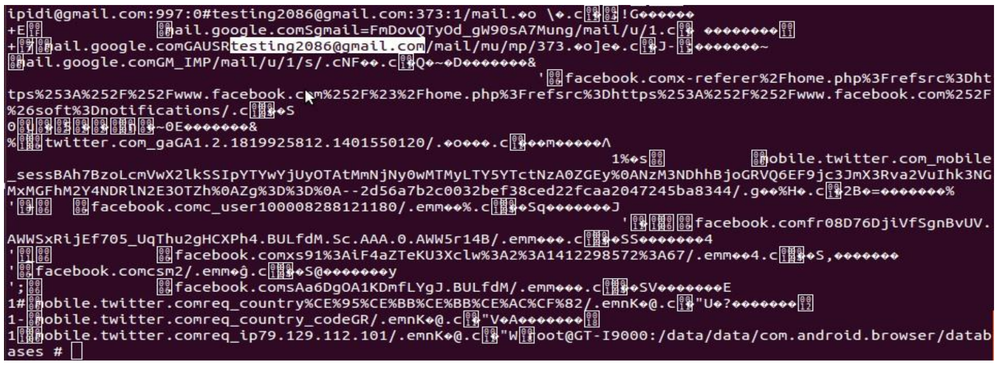
- **Default Android browser** (data/data/com.android.browser/databases/webview.db):



```
�oot@GT-I9000:/data/data/com.android.browser/databases # cat webview.db
cat webview.db
SQLite format 3��

�
�


O-�#
�HSy�
i�"� ��tablepasswordpassword  CREATE TABLE password (_id INTEGER PRIMARY KEY, host TEXT, username TEXT,
password TEXT, UNIQUE (host, username) ON CONFLICT REPLACE)�� �ndexsqlite_autoindex_password_1password
�+� �)tablehttpauthhttpauthCREATE TABLE httpauth (_id INTEGER PRIMARY KEY, host TEXT, realm TEXT, userna
me TEXT, password TEXT, UNIQUE (host, realm) ON CONFLICT REPLACE)/�� �ndexsqlite_autoindex_httpauth_1ht
tpaut�"� ��tableformdataformdataCREATE TABLE formdata (_id INTEGER PRIMARY KEY, urlid INTEGER, name TEX
T, value TEXT, UNIQUE (urlid, name, value) ON CONFLICT IGNORE)/�� �ndexsqlite_autoindex_formdata_1formd
ata��R���tableformurlformurl�CREATE TABLE formurl (_id INTEGER PRIMARY KEY, url TEXT)W��-�tableandr
 �L -�https://accounts.google.com/AccountChooser@��https://accounts.google.com/b/1/AccountRecoveryOption
sPrompt)��https://accounts.google.com/AddSession��https://m.facebook.com/)��https://mobile.twitter   7
��� ��7�mailtesting2086@gmail.com  ��7�emailtesting2086@gmail.com"�   7usernametesting2086@gmail.com
���!��7�mailtesting2086@gmail.com� ��7�emailtesting2086@gmail.com�
�oot@GT-I9000:/data/data/com.android.browser/databases #
```

Figure 153

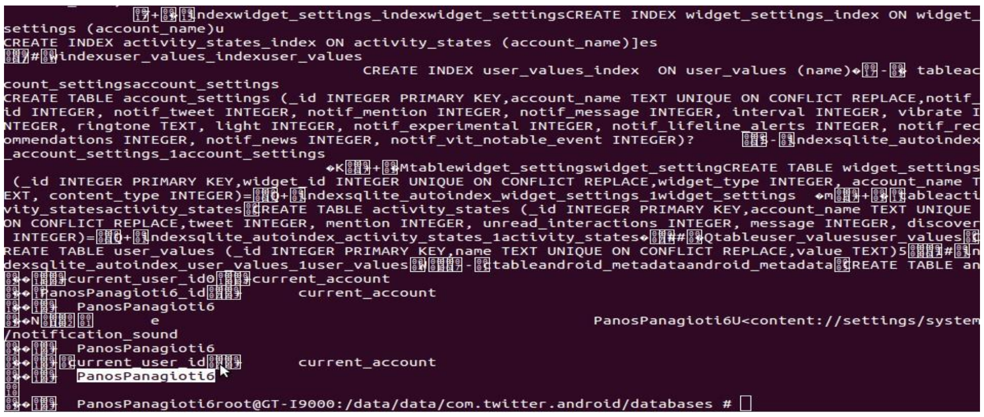
- **Default Android browser** (data/data/com.android.browser/databases/browser2.db):



```
ipidi@gmail.com:997:0#testing2086@gmail.com:373:1/mail.�o \�.c��!G������
+E�              �mail.google.comSgmail=FmDovQTyOd_gW90sA7Mung/mail/u/1.c� ��������
+�7�mail.google.comGAUSRtesting2086@gmail.com/mail/mu/mp/373.�o]e�.c�J-������~
�mail.google.comGM_IMP/mail/u/1/s/.cNF��.c�Q�~�D�������&
                                                          '�facebook.comx-referer%2Fhome.php%3Frefsrc%3Dht
tps%253A%252F%252Fwww.facebook.com%252F%23%2Fhome.php%3Frefsrc%3Dhttps%253A%252F%252Fwww.facebook.com%252F
%26soft%3Dnotifications/.c��S
0�����������~0E�������&
%�twitter.com_gaGA1.2.1819925812.1401550120/.�o���.c���m�����Λ
                                                                 1%�s�         �mobile.twitter.com_mobile
_sessBAh7BzoLcmVwX2lkSSIpYTYwYjUyOTAtMmNjNy0wMTMyLTY5YTctNzA0ZGEy%0ANzM3NDhhBjoGRVQ6EF9jc3JmX3Rva2VuIhk3NG
MxMGFhM2Y4NDRlN2E3OTZh%0AZg%3D%3D%0A--2d56a7b2c0032bef38ced22fcaa2047245ba8344/.g��%H�.c�2B�=�������%
'��  �facebook.comc_user100008288121180/.emm��%.c��Sq�������J
                                                                  '���facebook.comfr08D76DjiVfSgnBvUV.
AWWSxRijEf705_UqThu2gHCXPh4.BULfdM.Sc.AAA.0.AWW5r14B/.emm���.c��SS�������4
'��              �facebook.comxs91%3AiF4aZTeKU3Xclw%3A2%3A1412298572%3A67/.emm��4.c��S,�������
'�facebook.comcsm2/.emm�ĝ.c��S@�������y
';�              �facebook.comsAa6DgOA1KDmfLYgJ.BULfdM/.emm���.c��SV�������E
1#�mobile.twitter.comreq_country%CE%95%CE%BB%CE%BB%CE%AC%CF%82/.emnK�@.c�"U�?�������
1-�mobile.twitter.comreq_country_codeGR/.emnK�@.c�"V�A�������
1�mobile.twitter.comreq_ip79.129.112.101/.emnK�@.c�"W�oot@GT-I9000:/data/data/com.android.browser/datab
ases #
```

Figure 154

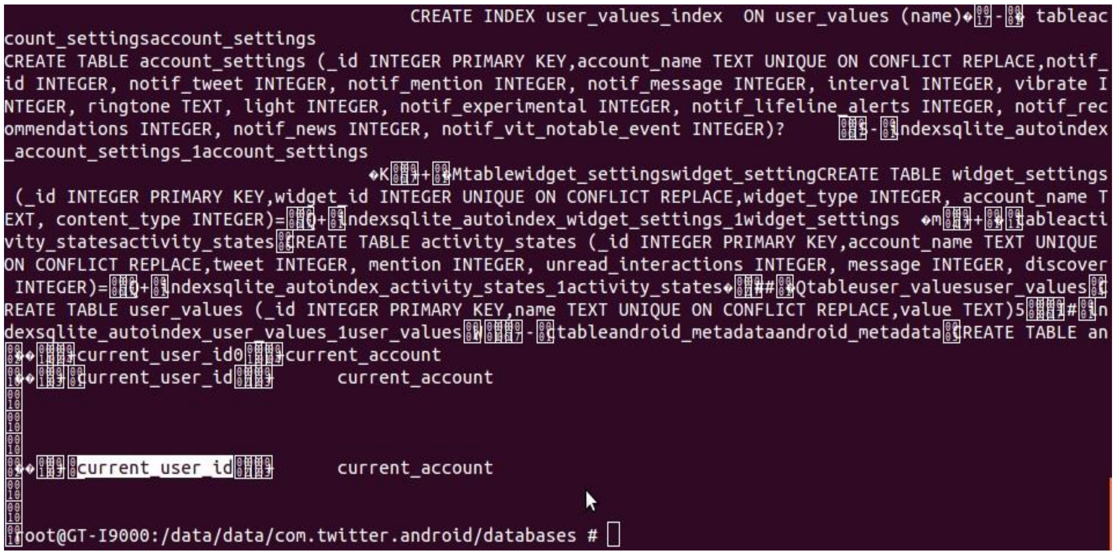
- **Twitter application** (data/data/com.twitter.android/databases/global.db) after log in to the application:



```
              �+��ndexwidget_settings_indexwidget_settingsCREATE INDEX widget_settings_index ON widget_
settings (account_name)u
CREATE INDEX activity_states_index ON activity_states (account_name)]es
�#�indexuser_values_indexuser_values
                                       CREATE INDEX user_values_index  ON user_values (name)��-� tableac
count_settingsaccount_settings
CREATE TABLE account_settings (_id INTEGER PRIMARY KEY,account_name TEXT UNIQUE ON CONFLICT REPLACE,notif_
id INTEGER, notif_tweet INTEGER, notif_mention INTEGER, notif_message INTEGER, interval INTEGER, vibrate I
NTEGER, ringtone TEXT, light INTEGER, notif_experimental INTEGER, notif_lifeline_alerts INTEGER, notif_rec
ommendations INTEGER, notif_news INTEGER, notif_vit_notable_event INTEGER)?     �-�ndexsqlite_autoindex
_account_settings_1account_settings
                                   �K�+�Mtablewidget_settingswidget_settingCREATE TABLE widget_settings
 (_id INTEGER PRIMARY KEY,widget_id INTEGER UNIQUE ON CONFLICT REPLACE,widget_type INTEGER, account_name T
EXT, content_type INTEGER)=�+�ndexsqlite_autoindex_widget_settings_1widget_settings  �m�+��ableacti
vity_statesactivity_states�CREATE TABLE activity_states (_id INTEGER PRIMARY KEY,account_name TEXT UNIQUE
ON CONFLICT REPLACE,tweet INTEGER, mention INTEGER, unread_interactions INTEGER, message INTEGER, discover
 INTEGER)=�+�ndexsqlite_autoindex_activity_states_1activity_states��#�Qtableuser_valuesuser_values�
REATE TABLE user_values (_id INTEGER PRIMARY KEY,name TEXT UNIQUE ON CONFLICT REPLACE,value TEXT)5�#�n
dexsqlite_autoindex_user_values_1user_values��-�tableandroid_metadataandroid_metadata�REATE TABLE an
��current_user_id0�current_account
���PanosPanagioti6_id�       current_account
���  PanosPanagioti6
��N��       e                                                   PanosPanagioti6U<content://settings/system
/notification_sound
���  PanosPanagioti6
����current_user_id�       current_account
���  PanosPanagioti6
�
���  PanosPanagioti6root@GT-I9000:/data/data/com.twitter.android/databases #
```

Figure 155

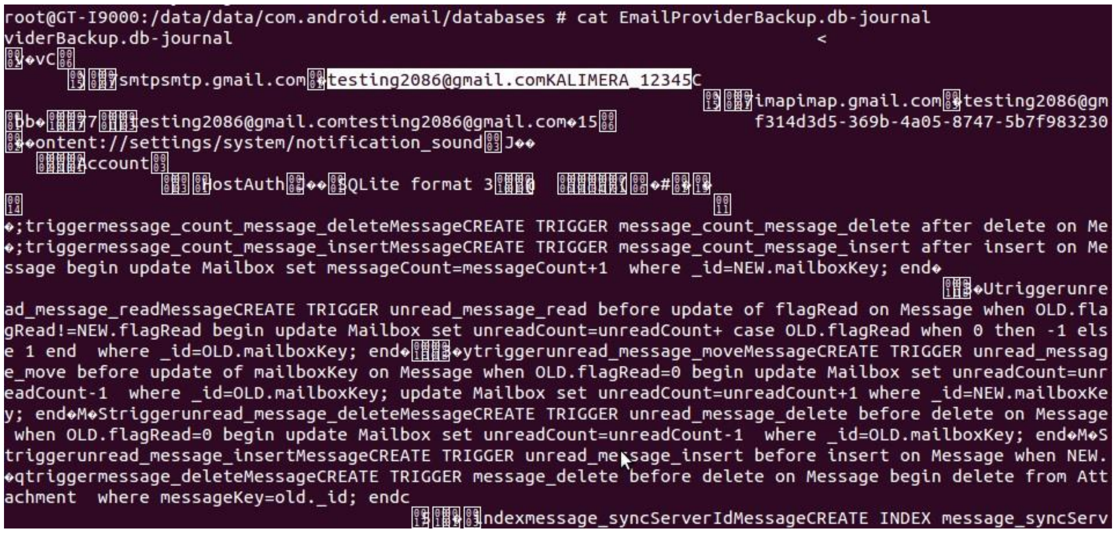
- **Twitter application** (data/data/com.twitter.android/databases/global.db) after sign out from the application :



```
                                        CREATE INDEX user_values_index  ON user_values (name)◆▯-▯◆ tableac
count_settingsaccount_settings
CREATE TABLE account_settings (_id INTEGER PRIMARY KEY,account_name TEXT UNIQUE ON CONFLICT REPLACE,notif_
id INTEGER, notif_tweet INTEGER, notif_mention INTEGER, notif_message INTEGER, interval INTEGER, vibrate I
NTEGER, ringtone TEXT, light INTEGER, notif_experimental INTEGER, notif_lifeline_alerts INTEGER, notif_rec
ommendations INTEGER, notif_news INTEGER, notif_vit_notable_event INTEGER)?     ▯▯-▯ndexsqlite_autoindex
_account_settings_1account_settings
                                    ◆K▯▯+▯◆Mtablewidget_settingswidget_settingCREATE TABLE widget_settings
 (_id INTEGER PRIMARY KEY,widget_id INTEGER UNIQUE ON CONFLICT REPLACE,widget_type INTEGER, account_name T
EXT, content_type INTEGER)=▯▯+▯ndexsqlite_autoindex_widget_settings_1widget_settings  ◆m▯▯+▯◆▯tableacti
vity_statesactivity_states▯REATE TABLE activity_states (_id INTEGER PRIMARY KEY,account_name TEXT UNIQUE
ON CONFLICT REPLACE,tweet INTEGER, mention INTEGER, unread_interactions INTEGER, message INTEGER, discover
 INTEGER)=▯▯+▯ndexsqlite_autoindex_activity_states_1activity_states◆▯▯#▯◆Qtableuser_valuesuser_values▯
REATE TABLE user_values (_id INTEGER PRIMARY KEY,name TEXT UNIQUE ON CONFLICT REPLACE,value TEXT)5▯▯#▯n
dexsqlite_autoindex_user_values_1user_values▯▯-▯tableandroid_metadataandroid_metadata▯REATE TABLE an
▯◆▯current_user_id0▯current_account
▯◆▯▯urrent_user_id▯       current_account
▯
▯
▯


▯◆▯▯current_user_id▯       current_account
▯
▯
▯
▯oot@GT-I9000:/data/data/com.twitter.android/databases # ▯
```

Figure 156

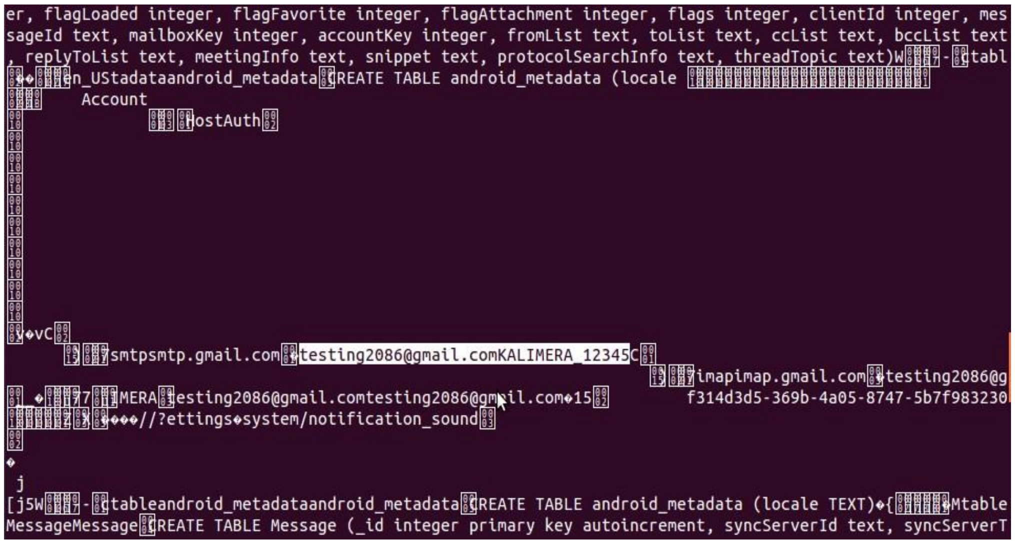
- **Default Android Email** (data/data/com.android.email/databases/EmailProviderBackup.db-journal):



```
root@GT-I9000:/data/data/com.android.email/databases # cat EmailProviderBackup.db-journal
viderBackup.db-journal                                                      <
▯◆vC▯
     ▯▯7smtpsmtp.gmail.com▯◆testing2086@gmail.comKALIMERA_12345C
                                                                   ▯▯7imapimap.gmail.com▯◆testing2086@gm
▯bb◆▯7▯testing2086@gmail.comtesting2086@gmail.com◆15▯         f314d3d5-369b-4a05-8747-5b7f983230
▯◆ontent://settings/system/notification_sound▯J◆◆
   ▯▯Account▯
                ▯▯HostAuth▯◆◆▯SQLite format 3▯0   ▯▯(▯◆#▯◆▯◆
▯                                                                  ▯
◆;triggermessage_count_message_deleteMessageCREATE TRIGGER message_count_message_delete after delete on Me
◆;triggermessage_count_message_insertMessageCREATE TRIGGER message_count_message_insert after insert on Me
ssage begin update Mailbox set messageCount=messageCount+1  where _id=NEW.mailboxKey; end◆
                                                                                           ▯◆Utriggerunre
ad_message_readMessageCREATE TRIGGER unread_message_read before update of flagRead on Message when OLD.fla
gRead!=NEW.flagRead begin update Mailbox set unreadCount=unreadCount+ case OLD.flagRead when 0 then -1 els
e 1 end  where _id=OLD.mailboxKey; end◆▯◆ytriggerunread_message_moveMessageCREATE TRIGGER unread_messag
e_move before update of mailboxKey on Message when OLD.flagRead=0 begin update Mailbox set unreadCount=unr
eadCount-1  where _id=OLD.mailboxKey; update Mailbox set unreadCount=unreadCount+1 where _id=NEW.mailboxKe
y; end◆M◆Striggerunread_message_deleteMessageCREATE TRIGGER unread_message_delete before delete on Message
 when OLD.flagRead=0 begin update Mailbox set unreadCount=unreadCount-1  where _id=OLD.mailboxKey; end◆M◆S
triggerunread_message_insertMessageCREATE TRIGGER unread_message_insert before insert on Message when NEW.
◆qtriggermessage_deleteMessageCREATE TRIGGER message_delete before delete on Message begin delete from Att
achment  where messageKey=old._id; endc
                                         ▯▯◆▯ndexmessage_syncServerIdMessageCREATE INDEX message_syncServ
```

Figure 157

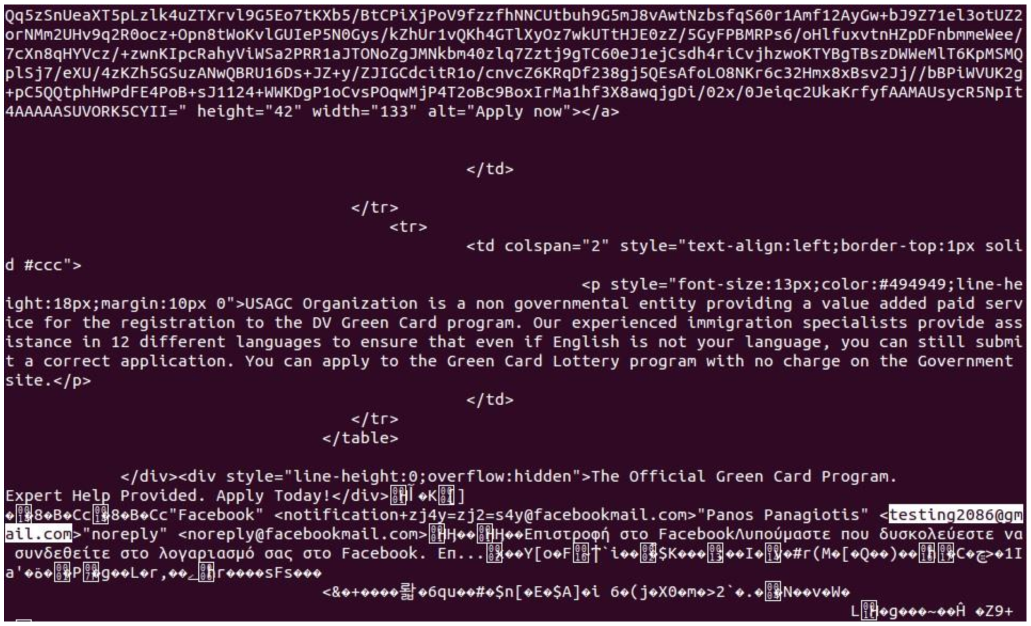
- **Default Android Email** (data/data/com.android.email/databases/emailProvider.db):



```
er, flagLoaded integer, flagFavorite integer, flagAttachment integer, flags integer, clientId integer, mes
sageId text, mailboxKey integer, accountKey integer, fromList text, toList text, ccList text, bccList text
, replyToList text, meetingInfo text, snippet text, protocolSearchInfo text, threadTopic text)W▯▯▯▯-▯tabl
▯◆▯▯en_UStadataandroid_metadata▯CREATE TABLE android_metadata (locale ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯     Account
▯             ▯▯ ▯HostAuth▯
▯
▯
▯
▯
▯
▯
▯
▯
▯
▯
▯◆vC▯
     ▯ ▯▯smtpsmtp.gmail.com▯◆testing2086@gmail.comKALIMERA_12345C▯
                                                               ▯ ▯▯imapimap.gmail.com▯◆testing2086@g
▯_◆▯▯▯77▯▯MERA▯testing2086@gmail.comtesting2086@gmail.com◆15▯       f314d3d5-369b-4a05-8747-5b7f983230
▯▯▯▯▯Z▯X▯◆◆◆◆//?ettings◆system/notification_sound▯
▯
◆
 j
[j5W▯▯▯-▯tableandroid_metadataandroid_metadata▯CREATE TABLE android_metadata (locale TEXT)◆{▯▯▯▯◆Mtable
MessageMessage▯CREATE TABLE Message (_id integer primary key autoincrement, syncServerId text, syncServerT
```

Figure 158

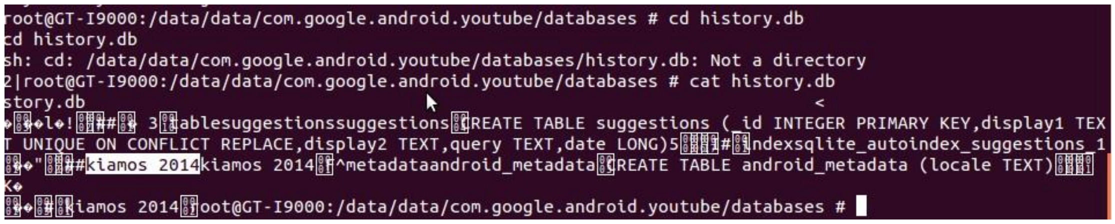
- **Gmail Application** (data/data/com.google.android.gm/databases/mailstore.testing2086@gmail.com.db):



```
Qq5zSnUeaXT5pLzlk4uZTXrvl9G5Eo7tKXb5/BtCPiXjPoV9fzzfhNNCUtbuh9G5mJ8vAwtNzbsfqS60r1Amf12AyGw+bJ9Z71el3otUZ2
orNMm2UHv9q2R0ocz+Opn8tWoKvlGUIeP5N0Gys/kZhUr1vQKh4GTlXyOz7wkUTtHJE0zZ/5GyFPBMRPs6/oHlfuxvtnHZpDFnbmmeWee/
7cXn8qHYVcz/+zwnKIpcRahyViWSa2PRR1aJTONoZgJMNkbm40zlq7Zztj9gTC60eJ1ejCsdh4riCvjhzwoKTYBgTBszDWWeMlT6KpMSMQ
plSj7/eXU/4zKZh5GSuzANwQBRU16Ds+JZ+y/ZJIGCdcitR1o/cnvcZ6KRqDf238gj5QEsAfoLO8NKr6c32Hmx8xBsv2Jj//bBPiWVUK2g
+pC5QQtphHwPdFE4PoB+sJ1124+WWKDgP1oCvsPOqwMjP4T2oBc9BoxIrMa1hf3X8awqjgDi/02x/0Jeiqc2UkaKrfyfAAMAUsycR5NpIt
4AAAAASUVORK5CYII=" height="42" width="133" alt="Apply now"></a>


                                                </td>

                                    </tr>
                                        <tr>
                                                <td colspan="2" style="text-align:left;border-top:1px soli
d #ccc">
                                                            <p style="font-size:13px;color:#494949;line-he
ight:18px;margin:10px 0">USAGC Organization is a non governmental entity providing a value added paid serv
ice for the registration to the DV Green Card program. Our experienced immigration specialists provide ass
istance in 12 different languages to ensure that even if English is not your language, you can still submi
t a correct application. You can apply to the Green Card Lottery program with no charge on the Government
site.</p>
                                                </td>
                                    </tr>
                                 </table>

            </div><div style="line-height:0;overflow:hidden">The Official Green Card Program.
Expert Help Provided. Apply Today!</div>▯Ĩ ◆K▯]
◆▯8◆B◆Cc▯8◆B◆Cc"Facebook" <notification+zj4y=zj2=s4y@facebookmail.com>"Panos Panagiotis" <testing2086@gm
ail.com>"noreply" <noreply@facebookmail.com>▯Ң◆◆▯Ң◆◆Επιστροφή στο FacebookΛυπούμαστε που δυσκολεύεστε να
 συνδεθείτε στο λογαριασμό σας στο Facebook. Επ...▯◆◆Y[o◆F▯†`i◆◆▯$K◆◆◆▯◆◆I◆▯◆#r(M◆[◆Q◆◆)◆◆▯▯C◆₢>◆1I
a'◆ö◆▯P▯g◆◆L◆r,◆◆_▯r◆◆◆◆sFs◆◆◆
                                 <&◆+◆◆◆◆뢌◆6qu◆◆#◆$n[◆E◆$A]◆i 6◆(j◆X0◆m◆>2`◆.◆▯N◆◆v◆W◆
                                                                                       L▯◆g◆◆◆~◆◆Ĥ ◆Z9+
```

Figure 159

- **Youtube application** (data/data/com.google.android.youtube/databases/history.db):



```
root@GT-I9000:/data/data/com.google.android.youtube/databases # cd history.db
cd history.db
sh: cd: /data/data/com.google.android.youtube/databases/history.db: Not a directory
2|root@GT-I9000:/data/data/com.google.android.youtube/databases # cat history.db
story.db                                                                  <
�▯�l�!▯▯##▯� 3▯ablesuggestionssuggestions▯CREATE TABLE suggestions (_id INTEGER PRIMARY KEY,display1 TEX
T UNIQUE ON CONFLICT REPLACE,display2 TEXT,query TEXT,date LONG)5▯▯▯▯#▯ndexsqlite_autoindex_suggestions_1
▯�"▯▯#kiamos 2014kiamos 2014▯f^metadataandroid_metadata▯CREATE TABLE android_metadata (locale TEXT)▯▯▯
K�
▯�▯#▯kiamos 2014▯oot@GT-I9000:/data/data/com.google.android.youtube/databases # █
```

Figure 160

5.2.2.2.4) Summary

Linux Memory Extractor tool (LiME) is considered as a special Mobile Forensics tool which belongs in the category of free tools. It can be operated either on Linux either on mobile device (running Android OS). In the second case LiME can acquire easily the memory either over TCP (like our experiment) either on SD Card (disk). The installation (of LiME) on the mobile device is characterized extremely complex (as it is required from the device to meet several requirements) However, after the installation the performance is highly simplified. Following the same methodology as the previous tool (Oxygen Forensic Suite 2014) makes extract entire ROM memory to a file, which with specific commands and keywords we try and retrieve the credentials of the user from various applications of the ROM.

In conclusion, the tool can successfully recover information as credentials (usernames and passwords) of all users who have used the mobile device before the extraction of data.

6) Conclusion

As time goes by we can observe that the field of Digital Forensics is becoming of growing interest in the IT world. The advancement in this field is so radical that it has extended into the area of mobile devices. Incidents of personal data breach are occurring more and more and therefore the need to improve digital system security is imperative.

At the same time, Androids in turn are becoming more popular as far as Smartphone devices are concerned and are rightfully one of the major operating systems. For the purposes of this research a specific Mobile Device with the above operating system was chosen in order to test the recoverability of specific data by default applications with the implementation of three scenarios.

The first scenario includes a Mobile Forensic non-commercial application (Root Browser), the second a Mobile Forensic commercial tool (Oxygen Suite) and the third Mobile Forensic non-commercial tool (Linux Memory Extrator Tool-LiME). In all three cases it became feasible to recover specific type of data (Credentials-Username and Password) from the user.

In conclusion, the recovery of personal and non-personal data from mobile devices using specialized technology and know-how can become feasible. But no one can ignore the ethical aspects and the clear distinction between "recovery" and "violation" (hacking), which are essentially two sides of the same coin.

7) Future Work

The security of personal data (credentials - Username & Password) on both mobile devices and non mobile devices will become the primary goal for users in the future. Therefore this is the topic I dealt with in my paper.

The creation of an application, which allows the individual user or programmer to manage the RAM memory in order to initialize the data and prevent any possible breach and recovery might be the solution to remedy this violation. The difficulty however that might occur, is the peculiarity of -Java- programming language, which is used in the majority of the applications. For this reason the implementation of a low level language (programming) is suggested or even the implementation (of the application) of a lower security level.

An application which is similar to the above proposal is the C-Cleaner. This option only refers to pieces related to credentials (Username & Password) or any files (e.g. logs) generated in the files of each application during the entry (log-in) of the user like sessions, tokens and various others.

The implementation of the proposal requires both experience and knowledge of the storage details (location and type of data) of the rest of the applications as the target clearance of credentials of the memory locations is required.

Appendix

References

URLS:

1) http://en.wikipedia.org/wiki/Android_(operating_system)

2) http://en.wikipedia.org/wiki/IOS

3) http://en.wikipedia.org/wiki/Windows_Mobile

4) http://en.wikipedia.org/wiki/Windows_Phone

5) http://en.wikipedia.org/wiki/BlackBerry

6) http://en.wikipedia.org/wiki/Bada

7) http://en.wikipedia.org/wiki/Symbian

8) http://en.wikipedia.org/wiki/Palm_OS

9) http://en.wikipedia.org/wiki/Firefox_OS

10) http://en.wikipedia.org/wiki/Sailfish_OS

11) http://en.wikipedia.org/wiki/Tizen

12) http://en.wikipedia.org/wiki/Mobile_device

13) http://en.wikipedia.org/wiki/Rooting_(Android_OS)

14) http://www.oxygen-forensic.com/en/

15) http://www.computersecuritystudent.com/FORENSICS/LIME/lesson1/index.html

16) http://en.wikipedia.org/wiki/Data_acquisition

17) http://dtiglobal.com/services/discovery-review-services/data-acquisition-and-forensics/

18) http://www.securestate.com/Services/Incident%20Response/Pages/Forensic-Acquisition.aspx

19) http://www.merrillcorp.com/data-aquisition-and-processing.htm

20) http://www.computersecuritystudent.com/FORENSICS/LIME/lesson1/index.html

21) http://www.xda-developers.com/lime-forensics-kernel-module-for-raw-memory-snapshots/

22) https://packages.debian.org/fr/sid/lime-forensics-dkms

23) https://www.guidancesoftware.com/solutions/Pages/by-task/investigations/acquire-data-from-a-smartphone.aspx?cmpid=nav

24) http://www.mccdaq.com/appnotes/android-data-acquisition-apps.aspx

25)http://www.academia.edu/4846019/Using_mobile_devices_for_data_acquisition_in_the_case_of_an_military_vehicle_team_in_fight_mission

26) http://www.tec-it.com/en/software/android/getblue/android-smartphone/Default.aspx

E-Books:

1) http://www.aueb.gr/users/amylonas/docs/secrypt11.pdf

2) http://link.springer.com/chapter/10.1007/978-3-642-21793-7_35

3) http://www.cs.kent.ac.uk/pubs/ug/2007/co620-projects/forensic/report.pdf

4) https://www.fbi.h-da.de/fileadmin/personal/h.baier/Lectures-winter-11/WS-11-Forensics/vorlesung_forensik_ws11-12_kap04-securing-phase-handout.pdf

5) http://vcgi.vermont.gov/sites/vcgi/files/training/chapter_6.pdf

6) http://seafloor.otterlabs.org/taskforce/pdf_2_web/4acquisition.pdf

7) http://www.academicjournals.org/article/article1379757650_%C3%96mer%20and%20Ceylan.pdf

8) https://lime-forensics.googlecode.com/files/LiME_Documentation_1.1.pdf

9) http://esatjournals.org/Volumes/IJRET/2014V03/I06/IJRET20140306030.pdf

10) http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1102&context=adf

11) http://www.aueb.gr/users/amylonas/docs/cosef.pdf

12) http://www2.hawaii.edu/~lipyeow/pub/dapd12-acqua.pdf

Books:

1) Digital Forensics for Network, Internet, and Cloud Computing: A Forensic Evidence Guide for Moving Targets and Data (Terrence V. Lillard, 2010)

2) Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet (Eoghan Casey, 2011),

3) The Best Damn Cybercrime and Digital Forensics Book Period (Jack Wiles and Anthony Reyes, 2011),

4) Handbook of Digital Forensics and Investigation (Eoghan Casey, 2009),

5) A Formalization of Digital Forensics (Ryan Leigland from University of Idaho, 2004),

6) Digital Forensics with Open Source Tools: Using Open Source Platform Tools: Using Open Source Platform Tools for Performing Computer Forensics on TargetSystems: Windows, Mac, Linux, Unix, etc (Google eBook) (Cory Altheide and Harlan Carvey, 2011),

7) Digital forensics of the physical memory (Mariusz Burdach, 2005),

8) Digital Crime and Forensic Science in Cyberspace (Kanellis Panagiotis, Kiountouzis Euaggelos, Kolokotronis Nicholas and Martakos Drakoulis, 2006)