The growth of Android, and its impact on Forensic Examiners

A minor thesis submitted in part fulfilment of the degree of M.Sc.

In Forensic Computing and Cyber Crime Investigation

With the supervision of Dr. Tahar Kechadi

Darragh Merrick

August 2013

Abstract

The sharp rise in smartphone sales on the consumer market caused a demand for forensic examination of the devices, which could not be met by existing computer forensics techniques. Mobile smartphones and tablets are becoming the most popular communications devices used at home and in the office, due to size, portability, improved battery life and easy to use applications and instant mail, social media etc. Extracting data from mobile smartphones and tablets in order to conduct forensic investigations can prove to be a difficult task. Computer Forensics methods and tools used on PCs, laptops or Macs are not effective on smartphones, due to differences in filesystems, storage multimedia and operating systems. The quality of investigation tools for mobile devices is not at the same level as tools available for desktops. Methods and tools are needed to enable forensic examiners to carry out "physical" and "logical" acquisition of data, such as operating system files, device memory and other technical information, plus personal email, documents or phone data.

The way that Android manufacturers have fragmented the operating system is a factor and on the Apple iOS the security is so effective that bypassing the PIN is a challenge for investigators. This comes at a time when both corporate examiners and law enforcement, who conduct this forensics work to get accurate, complete images from mobile device investigations, which will hold up under legal scrutiny. The BYOD (Bring Your Own Device) trend, in which employees use their own mobile devices at work, has caused a lack of data control and security policies on these devices, making forensics work more complicated. As a result of these challenges, a wide variety of tools exist to extract evidence from mobile devices, but no one tool or method can acquire all the evidence from all devices. It is therefore recommended that forensic examiners, especially those wishing to qualify as expert witnesses in court, undergo extensive training in order to understand how each tool and method acquires evidence, how it maintains standards for forensic soundness and how it meets legal requirements such as the Daubert standard or Frye standard. (Messmer E, 2012 October)

The initial aim of this research project was to understand the layers of the Android system, its security measures and how it operates in order to successfully extract data using forensically sound methods. The choice of Forensic techniques depends largely on the state in which the phone is discovered in. Extraction may range from using an adb command to gain access to using JTAG to physically dump data from the device, which is a difficult and time consuming process. Live data forensics methods must also be used to conduct a thorough examination, as there are partitions that are mounted in RAM and there is critical information recoverable from RAM, that would be lost if the device were to be powered off.

The result from the development stage of the project produced a suite of extraction programs for physically and logically extracting data, detecting which filesystem is present and for unlocking the screen. These programs could be altered to facilitate any new filesystems or Android OS changes which may occur in the future. A Raspberry Pi mobile forensics kit was created and also the ability to forensically examiner an Android using another Android device. The following chapters will explain in detail the findings from research and results from testing and development

.

Acknowledgments

I would like to express my very great appreciation to Dr Tahar Kechadi for his helpful and constructive suggestions during the planning and development of this research work.

Also, I wish to thank my Fiancée Roxanne for her support and patience throughout my study.

Table of Contents

Chapter 5 – Tool Development

Chapter 6 Conclusions

List of Diagrams

Android Overview 1

1.1What is Android?

Android is an operating system (OS) developed by the Open Handset Alliance (OHA). The Alliance is a coalition of more than fifty mobile technology companies ranging from handset manufacturers and service providers to semi-conductor manufacturers and software developers. This includes Google, eBay, Acer, ARM, HTC, Intel, Sony, LG Electronics, Qualcomm, T-Mobile and Sprint. It is a comprehensive, open source platform designed for mobile devices. The Android software stack includes an operating system, middleware and key applications. The independent Android OS separates the hardware from the software that it runs. This allows for a larger number of devices to run the same applications as well as creating a standardised environment for developers and consumers. The stated goal of the OHA is to "accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience." (OHA, 2007)

Android powers hundreds of millions of mobile devices in more than one hundred and ninety countries around the world. It is the largest installed base of any mobile platform and is rapidly expanding. Every day it is estimated, one million users power up their Android devices for the first time searching for apps, games, and other digital content. It offers a high quality, less expensive mobile experience on a world-class platform for telephony, email, multimedia, internet browsing and applications. Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using Java Programming language. The Google Play store is an open marketplace for distributing applications instantly.

Android features include the Dalvik virtual Machine optimized for mobile devices. An integrated browser based on the open source Web Kit engine. Optimized graphics powered by a custom 2D graphics library, 3D graphics based on the OpenGL ES 1.0 specification. The SQLite database provides structured data storage. Media support for common audio, video and still image formats (MPEG4, H.264, MP3, AAC, AMR, PNG, GIF) GSM Telephony, Bluetooth, EDGE, 3G and Wi-Fi, Camera, GPS, compass and Accelerometer. Android's development environment includes a device emulator tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE. Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. (Lessard J. & Kessler G. 2010)

1.2 Forensic Challenges

Android forensics is quickly changing, with new phones being introduced into the market frequently. As a result, bringing new file systems, formats and multimedia support. Past trends have seen two new Android OS releases per year. No one forensic tool supports all phones and extracts all data. Depending on a phone's accessibility, a Logical or physical extraction can be performed, although the device may be protected by encryption and/or contain hidden/deleted data. The investigator may encounter problems with non-standard characters, foreign languages and security locks/ PIN protection. The investigator must be very careful and alter the phone's data as little as possible, due to legal challenges. The data's integrity must be acceptable as evidence in a legal court. Standards and acceptable extraction methods vary from country to country.

As the development of Android devices expands, an increased awareness of the data they possess will equally need to be established. Unfortunately, much of that awareness will come from cybercriminal organizations who realize that successful attacks against the platform will yield significant results as the devices contain enormous quantities of personal and business information. The solution to this threat requires a deep understanding of the platform not only from core Android developers and manufacturers but also from app developers and corporate security officers. More secure apps will prevent loss of sensitive information as well as strong policies that can be put in place by IT security managers. Finally, the appeal of Android is not specific to any particular country or region. Android is global and will impact individuals, corporations, and agencies throughout the world. (Garfinkel S. 2011)

1.3 Aims and Objectives

At present there are a growing number of open and closed source tools available. Some are restricted to law enforcement and security agencies. Closed source tools include ViaForensics ViaExtract, Cellebrite, XRY, Radio Tactics, Oxygen Forensic and MOBIL edit. Although these tools offer reliable data extraction and a degree of automation the closed source tools are costly. On the open source market ViaForensics released Santoku, a Linux distribution with a selection of tools, such as AFLogical Open Source Edition, Android Encryption Brute Force, APK Tool (for reverse engineering), SQLMap, Android 2.3.3, 3.2, and 4.0.3 Emulators, Eclipse IDE, DroidBox and Android SDK Manager On the Android SDK there is a selection of tools such as ddms, ADB, Fastboot, Apk Builder and it contains emulators, so that development can be carried out and tested without even having a physical device. The Sleuth Kit supports Ext 4 and FAT, but not Yaffs2. It has many well-known tools including fsstat, fls.mmls and the graphical front-end Autopsy

The aim of this research project is to understand the layers of the Android system, it's security measures and how it operates in order to successfully extract data using forensically sound methods. During this research tests will be conducted on Android devices to get a feel for the standard of tools available and to select the most effective tools to use. In order to acquire a physical image of an Android device, root access is needed. There are many ways to gain root access, some will alter the device more than others, which may be the difference between being acceptable in court or not. Also there is a high risk involved during the rooting process, which

may result in complete data loss. It is worth noting that rooting an Android device is 100% legal, but it may void the warranty and if infected by malware, allow it complete access the operating system. The purpose of the Dalvik Virtual Machine is to sandbox each app to contain malicious apps. It would be beneficial for investigators to develop a rooting method for at least one or all Android devices that does not alter the images. There are many rooting methods that will alter the device in different ways. A forensically sound approach is needed that will not alter the phone, allowing the cryptographic hash to remain unchanged before and after the imaging process. The phone needs to be rooted to acquire the image, so making changes is unavoidable, although it does not always lead to automatic disqualification of the evidence either. The rooting technique is a necessary approach under the circumstance of current technology to meet the needs of the investigation. The techniques and their impacts on the evidence can be explained and therefore may be admissible.

Common "rooting" techniques developed by users of Android phones to gain full access to their devices can be found on internet hacking sites. It is not recommended to follow the user-oriented tutorials for rooting devices as they require you to install files unnecessary to forensics and make significant changes to the device, sometimes erasing user data. There are more suitable rooting methods that can exploit a bug in Android bypassing a fake message with executable firmware code to run as root. Exploits can be used in ADB by blocking ADB from calling the user shell and leave it running as root shell. When you start an ADB client, the client first checks whether there is an ADB server process already running. When the connection is established, the final command is to disable root access. This exploit will make this command fail, so that the user gets logged on as root. This method is most favourable for forensics, but not always achievable. Another ADB exploit is a buffer-overrun condition which runs in root, to execute arbitrary code with its root access. Security changes in API 9 ended this exploit.

In order to contribute to the field of Android Forensics and the open source community this research aims to develop new tools and methods to analyse and automate data extraction from Android devices. Due to the diversity of devices and varied methods for rooting, it can take a lot of time to analyse the devices. Through the use of Perl /Java and shell scripting it is possible to script commands to string search, call Linux/Android commands and write results to SQL tables, so these method could be used to automate some of the forensic processes and determine file system. The script could attempt root access, string search for file system clues, attempt open adb, physical extraction, logical extraction, brute force attacks on swipe codes and passwords. If the device is powered off the challenge will be to obtain the salt stored in the footer and add it to the Encrypted Master Key, in order to break the encryption. The test phones to be used are Samsung Galaxy S2 with EXT 4 file system, HTC Wildfire S with YAFFS2 file system and Samsung Galaxy Note 2 with EXT 4 file system

Another project aim is to research Android malware, its growth and how an examiner can detect it. App development and script searching to write the results to SQL tables are methods an examiner could use. Once methods have been established for a device, they will be documented in an online Android Forensics Wiki, which could be added to by other Forensic Investigators. Access to the site could be restricted by VPN or password authentication.

1.4 Growth of Android

In 2003 a small company known as Android Inc. operated secretly, revealing only that it was working on software for mobile phones. Google acquired this company on 17th August 2005 for $50,000,000, making it a fully owned subsidiary of Google. Under the new ownership Andy Rubin (co-founder and former CEO of both Danger Inc., and Android Inc.) led a team to develop a mobile device platform powered by the Linux Kernel.

On 5th November 2007, the mobile phone market speculated that Google would release its own mobile phone in competition with Apple's iPhone, but instead Google announced it had been developing a whole new open-source operating system named Android aimed at mobile phones, which would be released under the Apache 2.0 license. On that same day the Open Handset Alliance (OHA) unveiled itself with a goal to develop open standards for mobile devices. The OHA was originally a consortium of 34 technology companies, including mobile phone manufacturers, mobile network operators, application developers and processor developers, such as Sony, LG, Samsung, HTC, Motorola, Texas Instruments, Intel, Broadcom, NVidia, eBay, T-Mobile and Telefónica. These companies teamed up with Google to ensure the Android project would be well supported and ultimately successful. (Markoff, J November 2007).

The first Android-based phone, the HTC Dream (also marketed as the T-Mobile G) was released on 22nd October 2008. Since then, the Android mobile platform has quickly risen to the most popular operating system in the world by early 2013. Android has expanded to the tablet market with the Samsung Galaxy Tab, Google Nexus, Asus Transformer, Kindle Fire and Motorola Xoom all adopting the Android OS. Despite the success with the consumers, forensic analysts and security engineers have struggled to keep up with the new applications, firmware updates, OS releases, flash filesystems and technologies and to develop tools for investigating these devices. (HOLSON L. & HELFT M. August, 2008)

In traditional computer forensics techniques the examiner can create an image of a disk through a write blocker, taking a cryptographic hash before and after imaging. The disk can be kept as evidence, while the forensic examination is carried out on the image. In mobile phone forensics there is no standard way to copy data out of a device and it is difficult to image mobile phones without changing them. Android phones store an enormous amount of personal data, which may contain valuable information for a criminal investigation. There are number of forensic tools evolving, both open source and closed source for extracting data from the Android devices, but with the creation of new applications and the rise in Malware on the Android platform, forensic examiners are challenged to keep up with a fast changing environment.

Although most of Android focus is on smartphones and now tablets, there are many more devices that currently or in the near future will run Android. Some examples include GPS Satellite Navigation devices, gaming devices, televisions, netbooks, and a wide variety of other consumer devices. Android will be present in a large percentage of investigations for both forensic analysts and security engineer for the foreseeable future, therefore a deep understanding of the platform and applications, combined with a large variety of examination tools is needed.

Android File Systems and Architecture 2

2.1 Android Platform Versions and features

Android was designed for touch screen smart phones and tablets to interact with the internet and telephony services. Online access is a core feature of any Android device, whether using Global System for Mobile (GSM) Communications, Code Division Multiple Access (CDMA) or wireless networks (Wi-Fi). Android devices are capable of sending and receiving phone calls, text messages, web browsing and VoIP. A touch screen is typically used to interact with the device but external peripherals such as Bluetooth keyboards or other buttons can be attached. Android handles a wide range of hardware components including cameras(Front and Back), video recorders, multi-touch screens, GPS, accelerated 2D and 3D graphics, Bluetooth, WiMax, near-field-communications (NFC), many carrier network standards, storage, multi-core CPUs, AM/FM radio, voice recorder, Geo-magnetic, Gyro, RGB Light and Media player (audio and video).Android also has built in support for a wide range of sensors including accelerometers, gyroscopes, ambient light, magnetometers, proximity, pressure (i.e. barometers) and thermometers. (Andrew Hoog 2011)

Google Play, formerly known as the Android Market gives devices the ability to download, install and update applications (apps). Apps can extend the functionality of the device and can be free-of-charge or paid applications. Apps can be a source of information for forensic analysts, when recovering data from Android devices. Storage on Android devices is divided between on-device storage using flash (NAND) memory and an external SD card that is portable and intended to store larger amounts of data. Newer Android devices use an external Secure Digital (SD) card combined with an Embedded Multimedia Card (eMMC) which provides the large storage space required by many users. These storage devices exist because the user's app data, typically stored in /data/data, is isolated for security and privacy reasons. For storage of multimedia such as songs, pictures, videos or other files, the SD card's large capacity FAT file system partitions solve that issue. Also, some apps do not understand the EXT4/yaffs2 filesystem and need to be stored on FAT 16/32. The sensitive user data remains protected, yet the larger and more portable files are accessible to the user.

The platform is a large factor in determining the features a device can support. The official Android platforms are each assigned an application programming interface (API) level and receive a code name. While some devices will never support the latest version of Android, many do eventually receive the update. However, from a forensics and security perspective, the older APIs cannot be ignored. Like any software, Android is improving over time. The Android version number itself partly tells the story of the software platform's major and minor releases. What is most important is the API level. Version numbers change all the time, sometimes because the APIs have changed, and other times because of minor bug fixes or performance improvements. As forensic examiners it is important to know the API level of the device, as it will determine which features the phone can support and when rooting the phone the methods vary between APIs. Some ADB exploits that work on earlier versions have been blocked. The diagram in Figure 1: Timeline of Android Releases shows the release time of each new API.

Figure 1: Timeline of Android Releases

The following information is extracted from FAQoid (December 2012). It shows the timeline of Android from the beginning and the improvements made with each API release.

Base - Android Version 1.0 – API 1

This first release was released in September 2008, but was not used in any commercially available device.

Base 1.1 - Android Version 1.1 - API 2

This minor update was the first release used in a commercial device, the T-Mobile G1, in October 2008.

Cupcake -Android Version 1.5- API 3

This was a major release that was first utilized by a number of manufacturers. It was made available in April 2009, and was code-named Cupcake by Google. Enhancements include:

- Camcorder support to record and watch videos.
- Ability to easily upload images and videos to Picasa and YouTube.
- A number of Bluetooth improvements.
- Widgets and folders can now be placed on the home screen.
- Animation on various screen transitions.
- On-screen keyboard with text-prediction.

Donut -Android Version 1.6 – API 4

The v1.6 Software Development Kit (SDK) was released in September 2009. This release was code-named Donut. Enhancements include:

- New camera, camcorder and photo gallery interfaces.
- Improved voice search and search experience.
- Support for higher screen resolutions.
- Google turn-by-turn navigation.
- Text to speech engine.
- Multi-touch gesture support.
- VPN (Virtual Private Network) support.

Éclair - Android Version 2.0.1 – API 6, 2.1.x – API 7

In October 2009, the 2.0 SDK was released, and updated in January 2010 with version 2.1. Few devices were released with v2.0, but v2.1 has been quite popular. These releases were both code-named Éclair. Enhancements include:

- Improved UI.

- Contact and Account improvements and synchronization.
- Email support for Exchange, supports multiple account aggregation.
- More camera improvements including flash, digital zoom, white balance, scene modes and macro focus.
- Improved virtual keyboard.
- Browser improvements including key functions of HTML5.
- Improved speed.
- Improvements to Calendar and Google Maps.
- Bluetooth 2.1 support and related Bluetooth features.
- Live wallpapers.

Froyo - Android Version 2.2.x – API 8

In May 2010 the 2.2 SDK was released. This release is code-named Froyo (for Frozen Yogurt). Enhancements include:

- Camera control improvements with more on-screen buttons.
- Tethering with up to 8 Wi-Fi hotspots or via USB connection.
- Multi-lingual keyboard support allows quick language switching.
- More performance improvements for faster app access and browser speedups.
- Bluetooth improvements including voice dialling, contract sharing, support for Bluetooth car and desk docks.
- Numerous enhancements for Microsoft's Exchange, such as remote wipe, calendar support, global address lists and improved security.
- New home screen tips widget.

Gingerbread - Android Version 2.3, 2.3.1 and 2.3.2 – API 9
- Android Version 2.3.3 and 2.3.4 – API 10

In December 2010 the 2.3 SDK was released. This release is code-named Gingerbread. There has also been a stream of minor updates and bug fixes to version 2.3 since its initial release, with the latest as version 2.3.7 in September 2011. Enhancements from the first 2.3 release include:

- UI refinements for simplicity and speed.
- Faster, more intuitive text input on the virtual keyboard.
- One-touch word selection and copy/paste.
- Improved power management and power usage status.
- Application status and ability to manually stop applications.
- Internet phone calling.
- Near-Field Communication (NFC) support to read NFC tags.
- New download manager..
- Multiple camera support (i.e. front and rear cameras).
- Support for barometer, gravity, gyroscope, linear acceleration and rotation vector sensors.

Honeycomb - Android Version 3.0.x – API 11

In February 2011, the 3.0 SDK was released. This release is code-named Honeycomb and is targeted at devices with larger screens such as tablets. Enhancements include:

- New UI optimized for tablets includes a new system bar, action bar, customizable home screens and recent apps list.
- Redesigned keyboard for faster more accurate entry.
- Improved text selection, copy and paste.
- Synchronize media files via USB without mounting a USB mass-storage device.
- Support for physical keyboards via Bluetooth or USB.
- Bluetooth tethering allows more devices to share the network connection.

- Updated applications for larger screens including browser, camera, gallery, contact and email.
- Multi-core processor support.
- High-performance 2D and 3D graphic support.

Honeycomb-MR1 - Android Version 3.1.x – API 12

In June 2011, Android 3.1 was released. This release is also code-named Honeycomb (like version 3.0) and is targeted at devices with larger screens such as tablets. Enhancements include:

- UI refinements to navigation and animations.
- USB devices and accessories supported, including mice, keyboards, digital cameras and more.
- Expanded recent apps list.
- Resizable Home screen widgets.
- Support for joysticks and gamepads.
- Improved Wi-Fi networking stability, including connection while the screen is off.
- Updated set of standard apps, including browser, gallery calendar, contacts and email.
- Enterprise support features.

Honeycomb-MR2 - Android Version 3.2 – API 13

In July 2011, Android 3.2 was released. This release is also code-named Honeycomb (like versions 3.1 and 3.0). Enhancements over prior versions include:

- Further enhancements for Tablets.
- Compatibility zoom for fixed-sized applications.
- Direct application access to SD card file system.

Extended screen API for managing different screen sizes

Ice-Cream Sandwich - Android Version 4.0, 4.0.1 and 4.0.2 – API 14
Ice-Cream Sandwich_MR1 – Android Version 4.0.3 and 4.0.4 – API 15

In late October 2011, version 4.0 was released. This release is code-named Ice-Cream Sandwich. It merges the 3.x tab centric design and the v2.x phone based design into a single version.

Major UI changes and enhancements include:

- Refined UI.
- Recent Apps selection.
- Home folders and favourites tray.
- Resizable Widgets.
- Lock screen actions.
- Quick Response for calls.
- Network data control.

Other areas that are new or improved include:

- Social network improvements.
- Unified calendar.
- Camera and Camcorder changes - face detection, image stabilization, snapshots while video recording, new gallery app with photo editor.
- Browser can get full desktop versions of web sites.
- Improved Email.

- NFC based sharing.
- Face Unlock.
- Wi-Fi-Direct support.

Jelly Bean - Android Version 4.1 and 4.1.1 – API 16

In July 2012, version 4.1 was released. This release is code-named Jelly Bean.
UI changes and enhancements include:

- Improved touch response and transitions.
- Expandable, actionable notifications.
- Adaptive keyboard and guesses.

Other areas that are new or improved include:

- Instant review of taken photos.
- External Braille input and output via USB.
- Improved Voice search.
- NFC based photo sharing.
- USB audio.
- Google Wallet.

Jelly Bean_MR1 - Android Version 4.2 and 4.2.2 – API 17

In October 2012, version 4.2 was announced. This release retains the prior release's code-name of Jelly Bean.

UI changes and enhancements include:

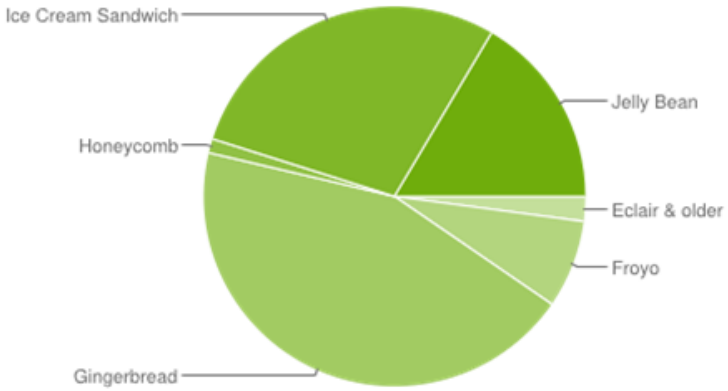
- Multiple users for tablets.

Other areas that are new or improved include:

- Photo Sphere to take 360 degree images.
- Built-in keyboard gesture typing (like Swipe).
- Daydream to display info while idle or docked.
- Beam photos and videos.

Current Distribution

The following pie chart and table is shows the number of Android devices that have accessed Google Play within a 14-day period ending on March 4, 2013, giving an indication of which API is currently most in use. It can be seen that API 10 is still most in use, API 15 second and API 16 third.



Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	1.9%
2.2	Froyo	8	7.5%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	43.9%
3.1	Honeycomb	12	0.3%
3.2		13	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	28.6%
4.1	Jelly Bean	16	14.9%
4.2		17	1.6%

Figure 2: Android Version Distribution Pie Chart

Android Developers, Dashboards (June 2013)

Google also released a graph displaying the historical distribution of Android versions for the seven-month period between 2 April 2009 and 2 April, 2013. The data are again based on devices accessing the Android Market but nicely displayed the progress of Android updates over time. At each point in time the colour portion is the percentage of devices running that OS. E.g. On 2/4/09 approximately 30% are running Cupcake, 52% running Donut and 18% running Éclair. Due to the fact that devices on older Operating Systems can't automatically upgrade to the new releases, trends show a slow increase in these OS's being adopted. It will take the lifetime of the phone to upgrade. However newer phones are upgradable. It can be seen on 2 April 2013 despite the new OS releases, Gingerbread still makes up 42% of current devices, Honeycomb did not distribute widely, while ICS was quiet popular and Jellybean popularity is increasing, due to upgrade ability of newer phones.

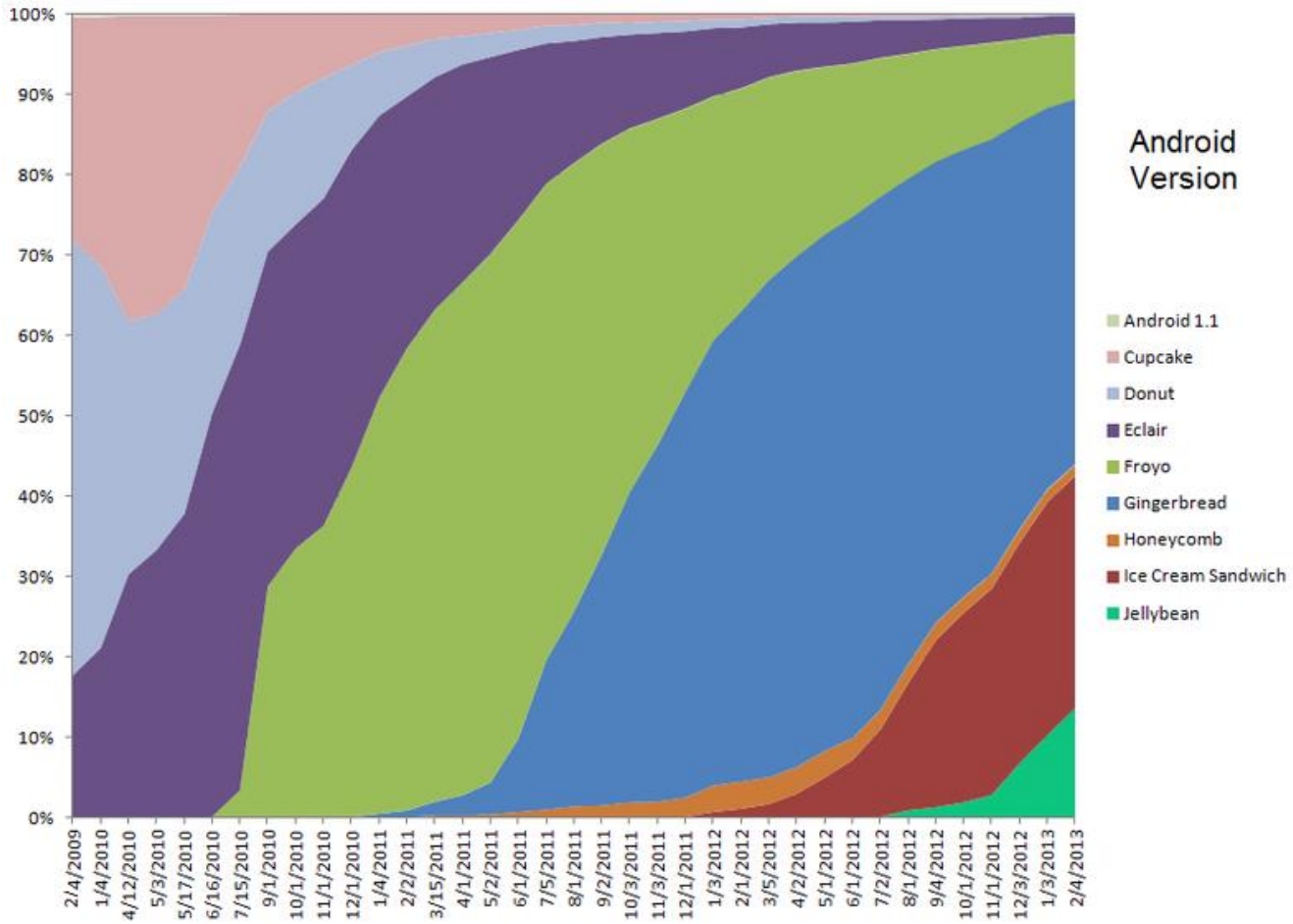


Figure 3 – Graph displaying the historical distribution of Android versions

https://commons.wikimedia.org/wiki/File:Android_historical_version_distribution.png

2.2 The Android Stack

The Android platform is best described as a stack because it is a collection of Components such as

- Linux kernel-based operating system
- Java programming environment
- Tool chain, including compiler, resource compiler, debugger, and Emulator.
- Dalvik VM for running applications

The Android OS can be referred to as a software stack of different layers. And the Layers are as follows;

- 1. Operating system.
- 2. Middle-ware.
- 3. Application.

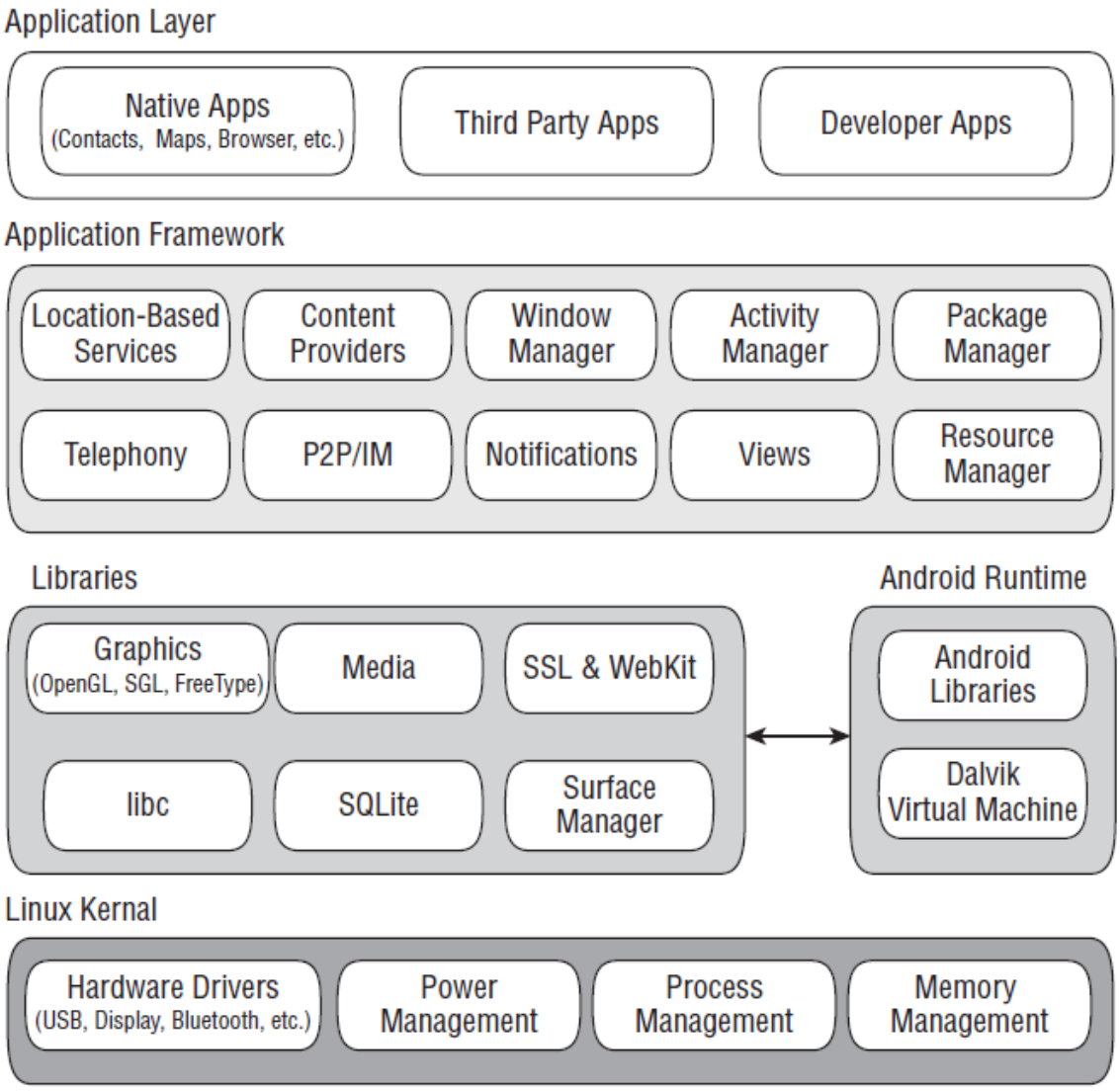


Figure 4 – The Android Software Stack

Samy M. (November 2010)

The Android software stack is composed of the elements shown in Figure 4 above. It consists of a Linux kernel and a collection of C/C++ libraries that are exposed through an application framework that provides services for and management run time applications

The elements of the Android software stack are:

1. Linux Kernel Core services (including hardware drivers, process and memory management, security, network, and power management) are handled by a Linux 2.6 kernel. The kernel also provides an abstraction layer between the hardware and the remainder of the stack.

2. Libraries Running on top of the kernel, Android include various C/C++ core libraries such as libc and SSL, as well as:

- A media library for playback of audio and video media
- A Surface manager to provide display management
- Graphics libraries that include SGL and OpenGL for 2D and 3D graphics
- SQLite for native database support
- SSL and Web-Kit for integrated web browser and Internet security

3. The Android run time is the engine that powers your android apps and along with the libraries, forms the basis for the application framework.

4. Core Libraries - While Android development is done in Java, Dalvik is not a Java VM. The core Android libraries provide most of the functionality available in the core Java libraries as well as the Android specific c libraries.

5. Dalvik Virtual Machine - Dalvik is a register-based virtual machine that's been optimized to ensure that a device can run multiple instances efficiently. It relies on the Linux kernel for threading and low-level memory management.

6. Application Framework - The application framework provides the classes used to create android apps. It also provides a generic abstraction for hardware access and manages the user interface and application resources.

7. Application Layer - All android apps, both native and third party, are built on the application layer using the same API libraries. The application layer runs within the Android run time using the classes and services made available from the application framework.

8. Android Application Architecture - Android architecture encourages component reuse allowing you to publish and share activities, services and data between applications with security restrictions defined by you. This enables developers to present new UI for out of the box components such as the phone dialler or contact manager, or adding new functionalists to them.(Samy M. November 2010)

The Android applications are controlled by:

- Activity Manager : Controls the life cycle of your activities, including management of the activity stack
- Content Providers : Lets your applications share data between android apps
- Notification Manager : Provides a consistent and non-intrusive mechanism for signalling your Users
- Resource Manager : Supports non-code resources like strings and graphics to be externalized

The VM nature of Android allows each application to run its own process. Security is permissions-based and attached at the process level by assigning user and group identifiers to the applications. Application cannot interfere with each other without being given the explicit permissions to do so. The security mechanisms of the Android OS could impede a forensic examination although some of the basic tools and techniques could allow investigators to recover data from the device. The first, most obvious step is to perform a traditional forensics analysis of the microSD card from the phone. This is the least effective method as it can only is access the data that apps directly store on the SD card. SD cards

use the FAT32 file system and are easily imaged and examined using traditional forensics tools (including write-blocking hardware).

2.3 Android Application Life Cycle

It is important for a forensic investigator to understand the Android Application Life Cycle, so that they are aware of the processes and activities that may be running on an Android device when it has been seized or recovered by Law Enforcement.

When a user navigates through, out of, and back to an app, the Activity instances in that app transition between different states in their lifecycle. For instance, when an activity starts for the first time, it comes to the foreground of the system and receives user focus. During this process, the Android system calls a series of lifecycle methods on the activity in which you set up the user interface and other components. If the user performs an action that starts another activity or switches to another app, the system calls another set of lifecycle methods on that activity as it moves into the background (where the activity is no longer visible, but the instance and its state remains intact).

The Android application life cycle is unique in that the system controls much of the life cycle of the application. All android apps, or Activities, are run within their own process. All of the running processes are watched by Android, depending on how the activity is running. Activities in the system are managed as an activity stack. When a new activity is started, it is placed on the top of the stack and becomes the running activity — the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

An activity has essentially four states:

1. If an activity in the foreground of the screen (at the top of the stack), it is active or running.

2. If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.

3. If an activity is completely obscured by another activity, it is stopped. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.

4. If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

The following diagram shows the important state paths of an Activity. The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states. The coloured ovals are major states the Activity can be in.

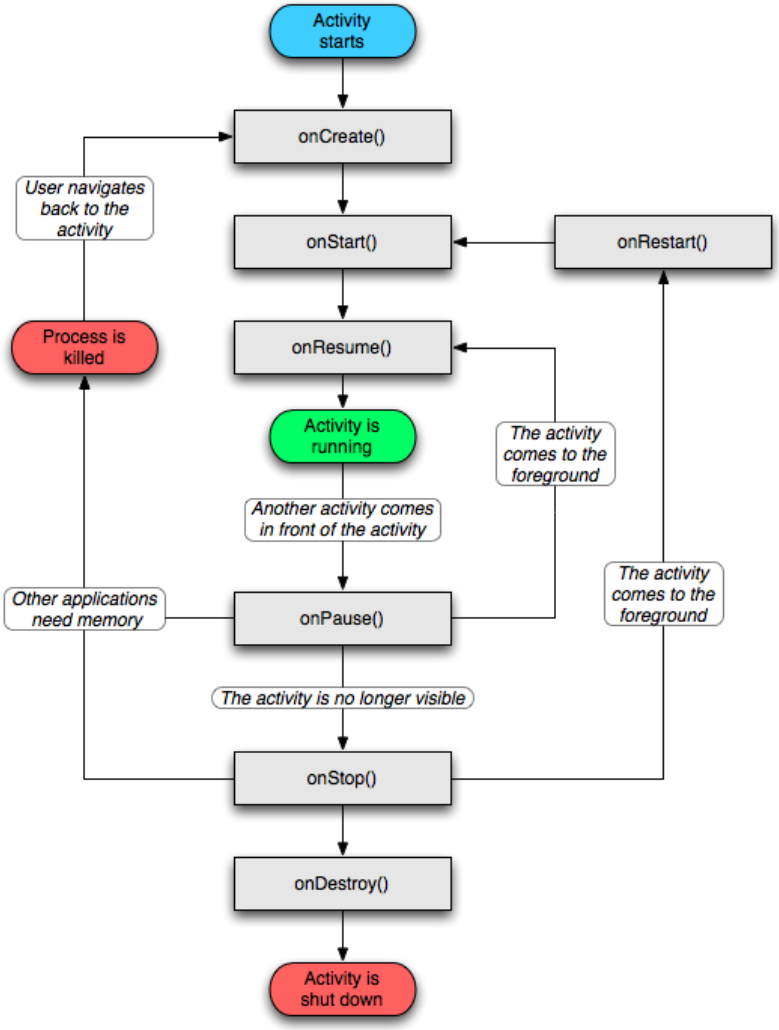


Figure 5 – The Android Activity Lifecycle
Android Developers (n.d.)

- onCreate() - called to set up the Java class for the instance of the android apps
- onStart() - technically, called to initiate the "visible" lifespan of the apps; at any time between onStart and onStop, the android application may be visible. We can either be onResume'd or onStop'ped from this state. Note that there is also an event for onRestart, which is called before onStart if the application is transitioning from onStop to onStart instead of being started from scratch.
- onResume() - technically, the start of the "foreground" lifespan of the app, but this does not mean that the app is fully visible and should be rendering – more on that later
- onPause() - the android apps is losing its fore grounded state; this is normally an indication that something is fully covering the app. On versions of Android before Honeycomb, once we returned from this callback, we could be killed at any time with no further app code called. We can either be onResume'd or onStop'ped from this state onStop() - the end of the current visible lifespan of the app – we may transition to
- onRestart() - to become visible again, or to onDestroy if we are shutting down entirely. Once we return from this callback, we can be killed at any time with no further app code called on any version of Android.
- onDestroy() - called when the Java class is about to be destroyed. (Android Developers n.d.)

2.4 The Android Boot Sequence

Android has been designed and built for the ARM architecture and therefore has some differences in how they initially start up and boot, compared to Desktop systems which are designed and built for the x86 platform.

The x86 platform is an instruction set for microprocessors based on the Intel 8086 CPU. It generally signifies that the OS uses the 32 bit version of the hardware architecture, but not all 32-bit processors are x86. There are some non x86 32-bit processors. The term x86 actually signifies backward compatibility with the original 8086 instruction set. The 32 bit x86, actually x86-32 became so popular that they were referenced as x86. The exact names would be x86-16, x86-32 and x86-64(or x64) for the x86 chips.

The ARM platform is designed for (Reduced Instruction Set Computer) RISC-based ARM processors, which require significantly fewer components that would typically be found in a traditional computer. The benefits are reduced costs, heat, and power for light, portable, battery-powered devices such as smart phones and tablet computers. At present the 32-bit instruction set architecture for ARM is mostly used.

The difference in platform between Android and Ubuntu/Windows desktops causes a difficulty for investigators when using forensic tools on the device. The tools built for the x86 platform will not be compatible with the ARM device. To tackle this problem a crosscompiler such Buildroot or CodeSourcery can be used to generate a cross-compilation toolchain that will allow the tools to run on the Android device. The boot-up of an Android system consists of several phases, which are outlined below;

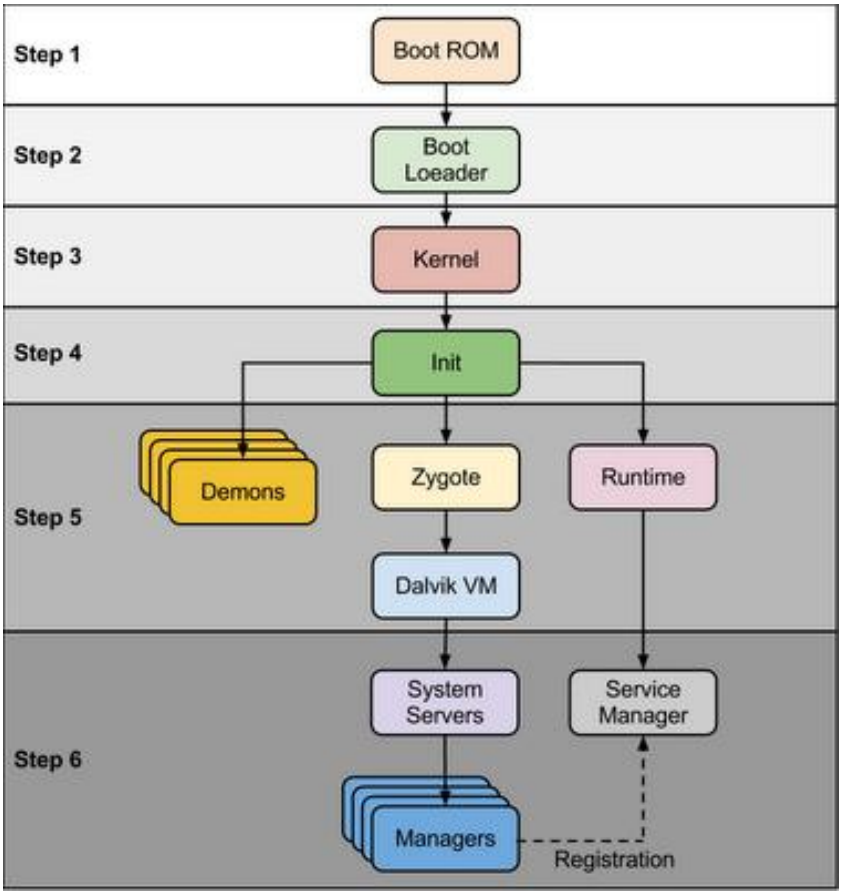


Figure 6 – The Android Boot Sequence

http://www.kpbird.com/2012/11/in-depth-android-boot-sequence-process.html

Step 1: Boot ROM

This is a small piece of code that is hardwired in the CPU ASIC (Application-specific integrated circuit) which loads x-loader. When power button is pressed, the Boot ROM code is executed from a pre-defined location. Bootloader is loaded into RAM and executed.

- (i)The Boot ROM code will detect the boot media using a system register that maps to the ASIC. This is to determine where to find the first stage of the boot loader.
- (ii) Once the boot media sequence is established the boot ROM will try to load the first stage boot loader to internal RAM. Once the boot loader is in place the boot ROM code will perform a jump and execution continues in the boot loader.

Step 2: Bootloader

X-loader is the initial Bootloader which initializes memory and loads u-boot from MMC, Nand or serial. The first program which runs on any Android system is the Bootloader. Technically, the Bootloader is independent of Android itself, and is used to do very low-level system initialization, before loading the Linux kernel. It detects if a special key is held, and can launch the recovery image, or the 'fastboot' Bootloader. The kernel then does the bulk of hardware, driver and file system initialization, before starting up the user-space programs and applications that make up Android. Often, the first-stage Bootloader will provide support for loading recovery images to the system flash, or performing other recovery, update, or debugging tasks. The Android Dev Phone is a SIM-unlocked and Bootloader unlocked device that is designed for advanced developers. The bootloader on the ADP 1 does not enforce signed system images. It is essential to have an unlocked bootloader to load custom images and to root devices. The Bootloader on the ADP detects certain key presses, which can be used to make it load a 'recovery' image (second instance of the kernel and system), or put the phone into a mode where the developer can perform development tasks ('fastboot' mode), such as re-writing flash images, directly downloading

Step 3: Kernel Android kernel start similar way as desktop Linux kernel starts, as kernel launch it start setup cache, protected memory, scheduling, loads drivers. When the kernel finishes the system setup first thing it looks for is “init” in system files and launch root process or first process of system.

- Once the memory management units and caches have been initialized the system will be able to use virtual memory and launch user space processes.
- (ii) The kernel will look in the root file system for the init process (found under system/core/init in the Android open source tree) and launch it as the initial user space process.

Step 4: 'init'

The init process started by the kernel parses init.rc. It starts several services and daemons including zygote. Zygote preloads classes which are shared between applications thus reducing the memory usage and application launch time, initializes Dalvic virtual vm and starts system server. The system server is the first java component to run in the system. It will start all the Android services.

- (i) The init process in Android will look for a file called init.rc. This is a script that describes the system services, file system and other parameters that need to be set up. The init.rc script is placed in system/core/rootdir in the Android open source project.

- (ii) The init process will parse the init script and launch the system service processes.

Step 5: Dalvik and Zygote

In a Java, separate Virtual Machine (VMs) instance will pop-up in memory for separate applications. For Android apps they need to launch as quick as possible, If the Android OS was to launch a different instance of Dalvik VM for every app, it would consume large quantities of memory and time. To overcome this problem the Android OS uses a system named “Zygote”. Zygote enables shared code across Dalvik VM, lower memory footprint and minimal start-up time. Zygote is a VM process that starts at system boot, then preloads and initializes core library classes.

The kernel runs /init:

- /init processes /init.rc and /init.<machine_name>.rc
- Dalvik VM is started (zygote).
- Several daemons are started:
- RILD - Radio Interface Link Daemon
- VOLD - Volume Daemon (media volumes for file systems, not related to audio volume)
- The system server starts, and initializes several core services such as telephony and Bluetooth.

Initialization is done in 2 steps:

- (i) The library is loaded to initialize interfaces to native services, then java-based core services are initialized in ServerThread::run() in SystemServer.java

  (ii) The activity manager starts core applications (which are Dalvik applications)

  - com.android.phone - phone application
  - android.process.acore - home (desktop) and a few core apps.

Other processes are also started by /init;

- adb
- mediaserver
- dbus-daemon
- akm

At this stage you can see the ‘Android” logo on device screen.

Step 6: System Services

After completion of the above steps, a runtime request is sent from Zygote to launch system servers. System Servers are written in both native and java. System servers run as a process. The same system server is available as System Services in Android SDK. System

servers contain all system services. Zygote now starts a new process to launch system services, the source code for this instruction can be found in ZygoteInit class and “startSystemServer” method.

Core Services started include;

- Starting Power Manager
- Creating Activity Manager
- Starting Telephony Registry
- Starting Package Manager
- Set Activity Manager Service as System Process
- Starting Context Manager
- Starting System Context Providers
- Starting Battery Service
- Starting Alarm Manager
- Starting Sensor Service
- Starting Window Manager
- Starting Bluetooth Service
- Starting Mount Service

Other services started include;

- Starting Status Bar Service
- Starting Hardware Service
- Starting NetStat Service
- Starting Connectivity Service
- Starting Notification Manager
- Starting DeviceStorageMonitor Service
- Starting Location Manager
- Starting Search Service
- Starting Clipboard Service
- Starting Check in Service
- Starting Wallpaper Service
- Starting Audio Service
- Starting HeadsetObserver
- Starting AdbSettingsObserver

Step 7: Boot Completed

Once the System Server is up and running and the system boot has completed there is a standard broadcast action called ACTION_BOOT_COMPLETED. To start your own service, register an alarm or otherwise make your application perform some action after boot you should register to receive this broadcast intent.

Android Blog (June 2009).

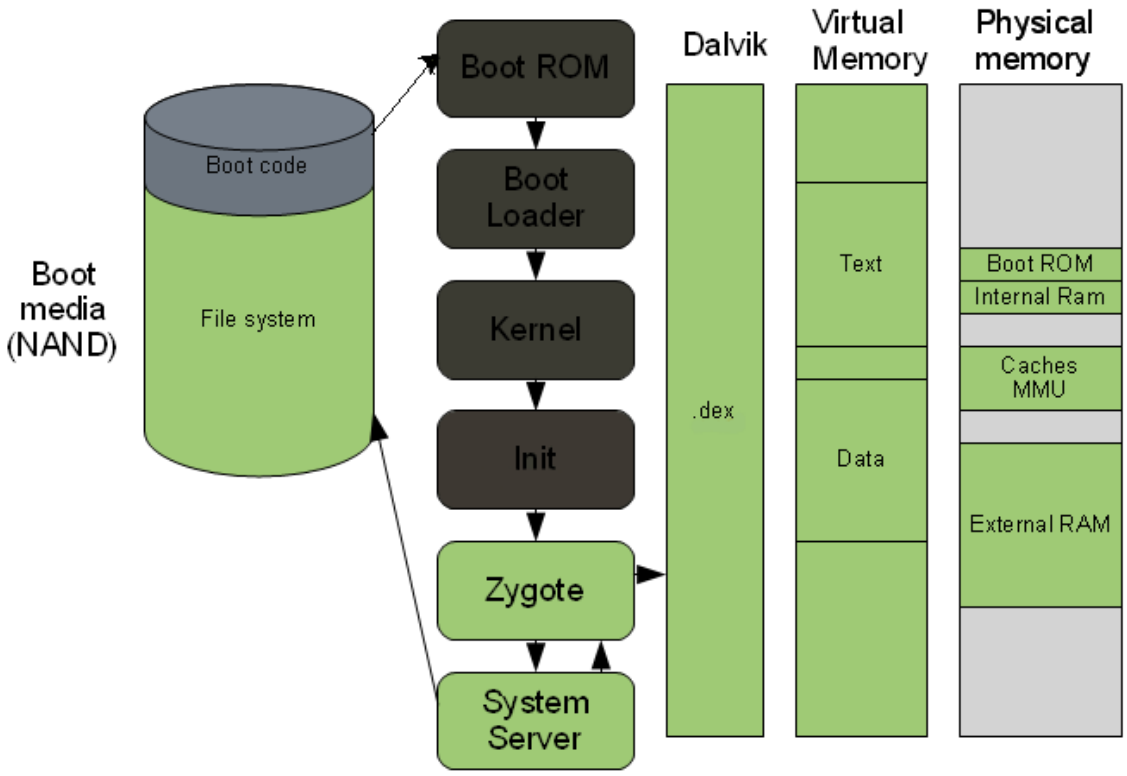


Figure 7 – End of Android Boot-up process

Android Blog (June 2009)

Summary of Boot-up Process

1. Execute Boot ROM code. This is stored in a hardware-specific area and keeps information on where to find the first stage of the boot loader, which is then loaded into RAM. You can compare the Boot ROM with the BIOS in a desktop PC.
2. Execute first stage of the boot loader. On desktop computers, this can be compared with the boot menu, e.g. Grub/LiLo on Linux. It loads its configuration at start-up and then turns control to the Linux Kernel.
3. The Linux Kernel together with the init process will initialize the base system, e.g. caches, file systems, etc., and then calls Zygote.
4. Zygote will initialize the Dalvik VM, and then starts the system server
5. The system server starts the Android-system, and sets up all Android-specific services, such as telephony manager and Bluetooth. Finally boot-up has completed.
6. The Boot completed event will be broadcast, so apps having registered listeners on this will be started.

Use the logcat command to analyze the Android Bootup;

```
dmerrick@ubuntu# adb logcat -d -b events | grep "boot"

        01-01 00:00:08.396 I/boot_progress_start(  754): 12559
        01-01 00:00:13.716 I/boot_progress_preload_start(  754): 17879
        01-01 00:00:24.380 I/boot_progress_preload_end(  754): 28546
        01-01 00:00:25.068 I/boot_progress_system_run(  768): 29230
        01-01 00:00:25.536 I/boot_progress_pms_start(  768): 29697
```

```
01-01 00:00:25.958 I/boot_progress_pms_system_scan_start(  768): 30117
01-01 00:00:40.005 I/boot_progress_pms_data_scan_start(  768): 44171
01-01 00:00:45.841 I/boot_progress_pms_scan_end(  768): 50006
01-01 00:00:46.341 I/boot_progress_pms_ready(  768): 50505
01-01 00:00:49.005 I/boot_progress_ams_ready(  768): 53166
01-01 00:00:52.630 I/boot_progress_enable_screen(  768): 56793

adb logcat -d | grep preload
10-15 00:00:17.748 I/Zygote  (  535): ...preloaded 1873 classes in 2438ms.
10-15 00:00:17.764 I/Zygote  (  535): ...preloaded 0 resources in 0ms.
10-15 00:00:17.772 I/Zygote  (  535): ...preloaded 15 resources in 7ms.
```

- Bootchart - see Using Bootchart on Android
- strace is pretty handy also, to see the timings for system calls from a process as it runs. Strace can be used as a wrapper for a program in init.rc, and save the results to a file
    - Use -f to follow sub-processes
    - Use -tt to get detailed timestamps for syscalls
    - Use -o to output the data to a file

Below is an example of using strace to follow the startup of zygote and the apps that are forked from it. Replace:

```
service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server
```

With;

```
service zygote /system/xbin/strace -f -tt -o /cache/debug/boot.strace /system/bin/
app_process  -Xzygote  /system/bin --zygote --start-system-server
```

2.5 Android File systems

The primary focus of forensic analysts and security engineers is to acquire, recover, analyse and understand data stored on a device, including what types of data are stored, where they are stored, how they are stored and characteristics of the physical mediums on which they are stored. Identifying the system partitions, filesystem, files, and other artefacts is vital before attempting to recover data from an Android device. Android is a combination of both well-known artefacts such as those found in Linux and other non-familiar, such as the Dalvik VM and the YAFFS2 file system.

The ‘Yet Another Flash File System 2’ (YAFFS2) filesystem has been adopted by many Android devices. YAFFS, developed in 2002, was the first file system designed for NAND flash memory devices. YAFFS2 was designed in 2004 in response to the availability of larger sized NAND flash devices; older chips support a 512 byte page size whereas newer NAND memory has 2096 byte pages. YAFFS2 is backward compatible with YAFFS. When Android 2.3 Gingerbread was released it was optimised for dual-core processing. A new wave of dual-core handsets which entered the market from Samsung, Motorola and LG adopted Ext4, the current Linux kernel file system.YAFFS2 is single-threaded and does not support devices with more than one processor, whereas Ext4 doesn't have this limitation. The new filesystem is more suited for usage with the multi-core ARM-based chipsets that will be standard in handsets and tablets over the coming years. Ext4 improves the handling of data loss, if developers make sure their application data is getting to persistent storage on time. There are newer Linux file systems out there, like Oracle's Btrfs which is a copy-on-write file system intended to address various weaknesses in current Linux file systems. Its primary focus points include fault tolerance, repair, and easy administration. It offers scalability and reliability, but it is still currently considered unstable.

The choice of file system is dependent on the storage technology: Android devices with raw NAND flash will continue to use yaffs2, but those using eMMC, which is basically a micro SD card soldered on to the main board, have to use a "normal" file system. In that case, ext4 seems a reasonable choice. Also bare Nand requires a lot of pins. The eMMC storage is essentially a serial interface, with variable width data path (8 bit max). So with eMMC, the required pin counts can be reduced, allowing for cheaper boards.

Google's Nexus S smartphone is the first Android device to use the Ext4 filesystem. Most Android devices currently use YAFFS, a lightweight filesystem that is optimized for flash storage and is commonly used in mobile and embedded devices. The problem with YAFFS is that it is single-threaded and will not support on dual-core systems. With the advancement in technology, combined with the increasing complexity and speed of devices, the need for more processing power will require dual-core and quad-core processors. Trends show that the majority of new devices are using duo-core and quad processors, which could cause YAFFS2 to become obsolete.

The table below compares the specifications of 3 Android devices with multi-core processors.

	HTC One X	HTC One X	Samsung Galaxy Nexus
CPU model	NVIDIA Tegra 3	Snapdragon S4 MSM8960	TI OMAP 4460
CPU instruction set	ARMv7	ARMv7	ARMv7
CPU architecture	ARM Cortex-A9	Qualcomm Krait	ARM Cortex-A9
Semiconductor technology	40nm	28nm	45nm
CPU speed	1.5 GHz	1.5 GHz	1.2 GHz
CPU cores	4	2	2
GPU model	GeForce	Adreno 225	PowerVR SGX540
Android version	4.0.3	4.0.3	4.0.2
Firmware version	1.28.401.9	1.73.502.1	ICL53F

Figure 8 – Android Multi-core Processor specifications

What Data is Stored on Android Devices

Android devices store an enormous amount of data, typically combining both personal and work data. Apps are the primary source of these data, and there are a number of sources for apps including:

- Apps that ship with Android
- Apps installed by the manufacturer
- Apps installed by the wireless carrier
- Additional Google/Android apps
- Apps installed by the user, typically from the Android Market

Common data found on Android devices includes the following;

- Text messages (SMS/MMS)
- Contacts
- Call logs
- E-mail messages (Gmail, Yahoo, Exchange)
- Instant Messenger/Chat
- GPS coordinates
- Photos/Videos

- Web history
- Search history
- Driving directions
- Facebook, Twitter and other social media clients
- Files stored on the device
- Music collections
- Calendar appointments
- Financial information
- Shopping history
- File sharing

App Data Storage Directory Structure

Android applications primarily store data in two locations, internal and external storage.. In the external data storage areas (the SD card and emulated SD cards), applications can store data in any location they wish. However, internal data storage is controlled by the Android APIs. When an application is installed (through either the market place or in the build shipped to the consumer), an internal data storage is saved in a subdirectory of /data/data/ named after the package name. For example, the default Android browser has a package name of com.android.browser and, as such, the data files are stored in /data/data/com.android.browser. Inside the applications /data/data subdirectory, there are a number of standard directories found in many applications as well as directories that developers control.

How Data is Stored

Android provides developers with five methods for storing data to a device. Forensic examiners can uncover data in at least four of the five formats. Therefore, it is important to understand each in detail. Persistent data are stored to either the NAND flash, the SD card, or the network. Android provides several options to save persistent application data. Where to store the data depends on whether it is be private to an application or accessible to other applications (and the user) and how much space its data requires. The data storage options are as follows:

- Shared Preferences - Stores private primitive data in key-value pairs.
- Internal Storage - Stores private data on the device memory.
- External Storage - Stores public data on the shared external storage.
- SQLite Databases - Stores structured data in a private database.
- Network Connection- Stores data on the web with your own network server.

Android provides a way to expose even your private data to other applications using a content provider. A content provider is an optional component that exposes read/write access to an application data, subject to whatever restrictions were accepted during its installation.

Shared Preferences

The Shared Preferences class provides a general framework that allows the user to save and retrieve persistent key-value pairs of primitive data types. Shared Preferences allow

the user to save any primitive data: Booleans, floats, integers and strings. This data will persist across user sessions (even if your application is killed).

Internal Storage

Files can be saved directly on the device's internal storage. By default, files saved to the internal storage are private to your application, meaning the user and other applications cannot access the files. When the user uninstalls the application, these files are removed.

External Storage

Every Android-compatible device supports a shared "external storage" that can be used to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

It's possible that a device using a partition of the internal storage for the external storage may also offer an SD card slot. In this case, the SD card is not part of the external storage and applications cannot access it (the extra storage is intended only for user-provided media that the system scans). Caution: External storage can become unavailable if the user mounts the external storage on a computer or removes the media, and there's no security enforced upon files saved to the external storage. All applications can read and write files placed on the external storage and the user can remove them.

SQLite Databases

Android provides full support for SQLite databases. Any databases created by a user will be accessible by name to any class in the application, but not outside the application. The Android SDK provides dedicated APIs that allow developers to use SQLite databases in their applications. The SQLite files are generally stored on the internal storage under /data/data/<packageName>/databases.

SQLite databases are a rich source of forensic data. The built-in Android browser, based on the WebKit Open Source Project provides a great example. In the subdirectories of /data/data/com.android.webkit there can be;
Change this for the phone example.

- app_icons: 1 database of web page icons
- app_cache: 1 database containing web application data cache
- app_geolocation: 2 databases relating to GPS position and permissions
- app_databases: 21 databases providing local database storage for supporting web sites
- databases: 3 databases for the browser and browser cache

There is very high potential of recovering forensically valuable data from these files.

Using a Network Connection

You can use the network (when it's available) to store and retrieve data on your own web-based services. To do network operations, use classes in the following packages:

- java.net.*
- android.net.*Dropbox, Box.net, Rapidshare and SugarSync are all examples of cloud based storage service (Android Developers n.d.)

Android Forensic Techniques 3

3.1 Introduction

Mobile Phone forensics is a branch of digital forensics, related to the examination and data extraction from mobile devices such as mobile phones, smartphones, handheld GPS, E-readers and tablet computers. Mobile phones, specifically smart phones, have dramatically increased in both world-wide use and functionality in recent years. Extracting data from mobile devices varies in methodology from Android phones, iPhones, and Blackberries. Examination of a mobile device includes:

- Identification of the device under investigation.
- Isolation from the network to prevent additional data from entering the device or execution of a remote wipe of the data.
- Extraction of the device's data in a sound and forensic format that will hold up in court.
- Validation of the steps and investigation methods used in the examination.
- Reporting on the findings in a complete and factual manner that is easy to understand.

Traditionally mobile phone forensics has been associated with recovering SMS/ MMS messaging, call logs, contact lists and phone IMEI, but since the introduction of smartphones this is no longer the case, as these devices are a hybrid of computer and telephone which may additionally contain video, email, web browsing information, application data, location information, and social networking messages and contacts. These devices have potential for criminal activity such as bank fraud over e-mail, gang communications, trafficking of child pornography, borderless VoIP telephony, and organisation of drug distribution or even issuing commands to Botnot C&C servers.

Evidence that can be recovered from a mobile device may come from several different sources. The data stored on these devices could prove vital as evidence in a criminal investigation, but it is difficult to recover the data in a forensically sound manner. In computer forensics, hard disks can be removed from a computer and connected to a Forensic workstation to be imaged offline through a write blocker. A cryptographic hash can be taken and forensic analysis can be carried out on the images to carve and recover data. Filesystems such as FAT16/32, NTFS or EXT2/3 are well supported by forensic tools.

The difficulty with imaging Android devices is that the device has to be rooted in order to extract a physical image. Due to the architecture of phones, it is not possible to acquire data when the device is powered down, therefore data extraction must be performed live. It is good practice when a device has been seized to plug it into a recharger and put it into a Faraday cage, to prevent remote wiping of the device, but by even simply doing this, the phone would recognise the network disconnection and change its status information, triggering the memory manager to write data.

Once the phone has been rooted the forensics tools such as busybox, dd, Nandump, AFlogical or Lime can be pushed to the device using an adb connection from a pc. Writing directly to suspect's device is not a forensically sound practice, but in order to extract the information it is unavoidable. To minimise the impact, the tools can be written into '/dev' directory which is mounted on the tmpfs partition and will be stored in RAM. Another obstacle is the variety of devices, operating systems, storage media types, forensic tools available and standardisation within the industry. The security mechanisms of the Android OS could impede a forensic investigation, although there are techniques to allow investigators to bypass

security locks and gain access to a device. Live data forensics can also be used when the device is turned on, using the Linux Memory Extractor (Lime) to extract data from tmpfs and to recover process information at time of recovery. If the device is turned off, the encryption will need to be cracked or the swipe lock/ PIN bypassed. This chapter outlines forensic extraction methods, using open source tools.

3.2 Choice of Forensic Techniques

In order for a forensic investigator to examine a smartphone, the device must be handled in accordance with ACPO Guidelines for Forensic Computing**.** The following guidelines from the Association of Chief Police Officers are the principles used during a forensic analysis of computer equipment or mobile phones.

Principle 1: No action taken by law enforcement agencies or their agents should change data held on a computer or storage media which may subsequently be relied upon in court.

Principle 2: In exceptional 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.

Principle 3: An audit trail or other record of all processes applied to computer based electronic evidence should be created and preserved. An independent third party should be able to examine those processes and achieve the same result.

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

This means that investigators must extract data required for evidence, without changing data on the smartphone, but without rooting the phone and installing extraction tools, the investigator has limited access to the device and can't extract a physical image. When an Android device is recovered by law enforcement, the state in which the device is found, will determine how the investigator must precede, in order to preserve as much data as possible and successfully extract data from the device. It is unavoidable to make changes to the device, but changes made must be minimal and well documented with valid reasons for why the actions were carried out. Important factors to consider are;

- Is the device powered on/off?
- Is the device PIN locked, Swipe locked or a combination of both?
- Is ADB on/off?
- Is the device rooted?
- Is the bootloader unlocked (S-on/S-off)?
- Is the file-system YAFFS2 or EXT4?

From these possibilities a flow chart has been devised showing how an investigator should precede, See Figure 9. The sections following explain in detail, each course of action.

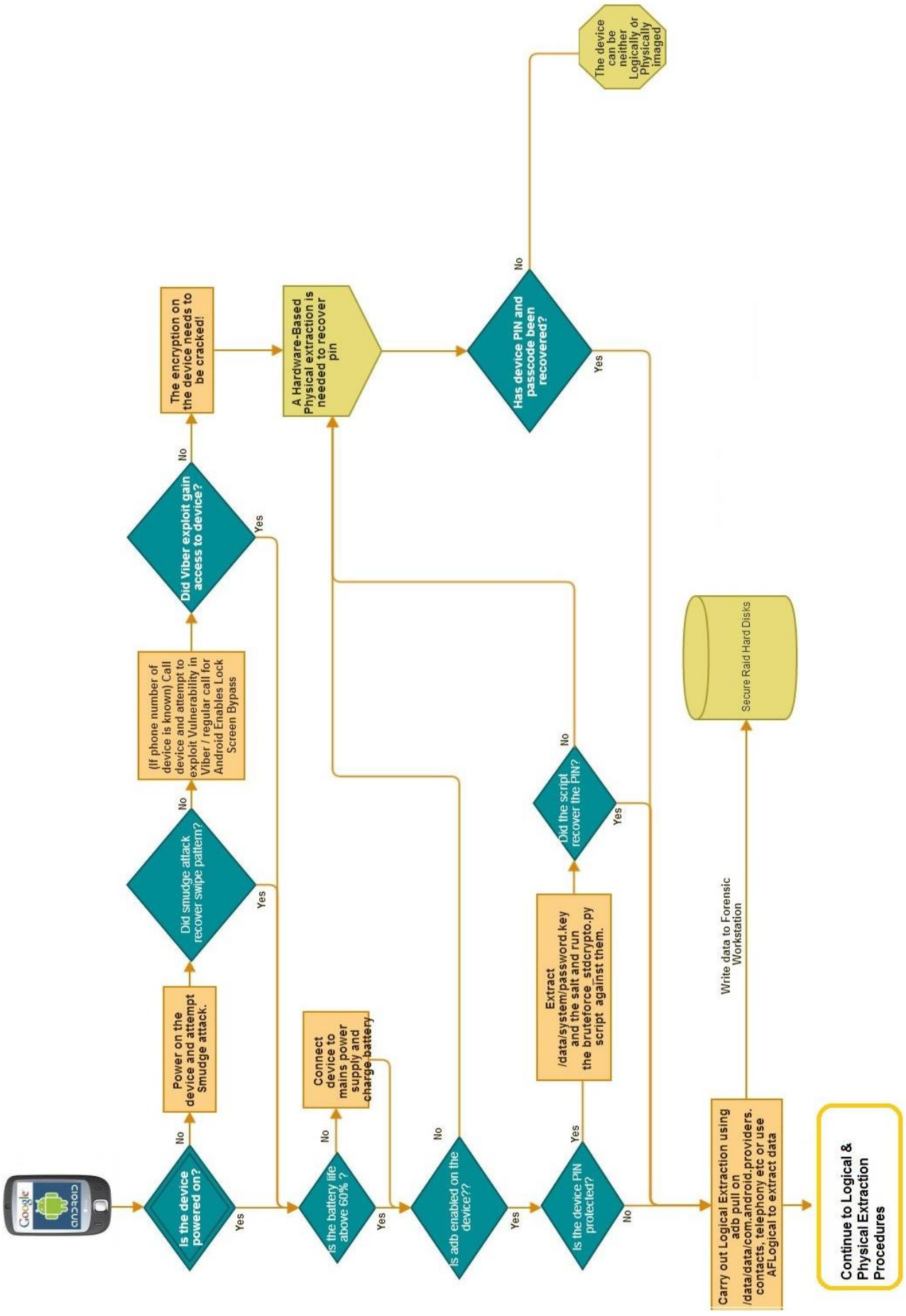


Figure 9(a) – Flowchart showing how to extract data from Android device

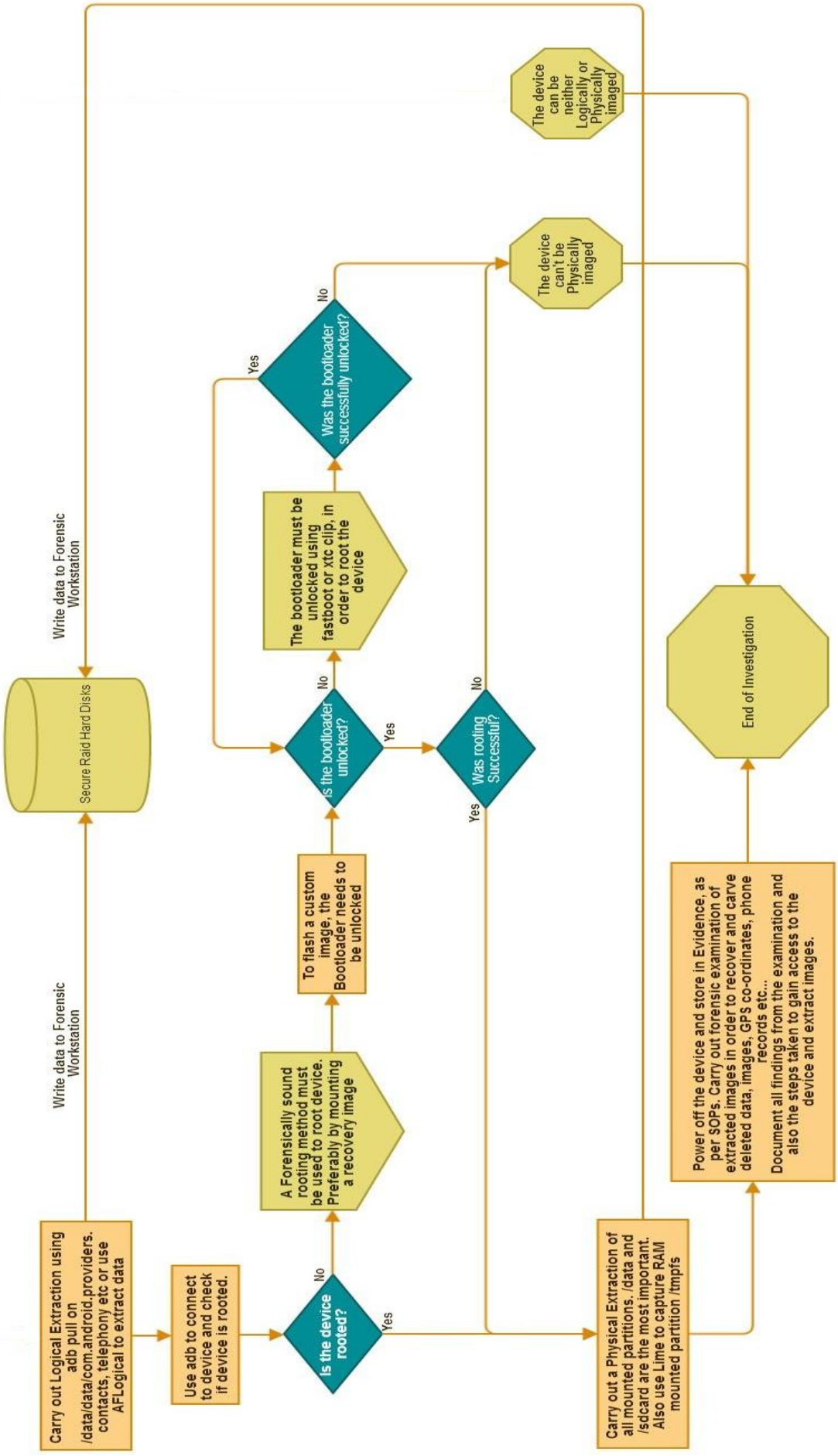


Figure 9(b) – Flowchart showing how to extract data from Android device

Smudge Attacks

If a smartphone is powered off or pattern locked, a smudge attack can be a very effective way of using residual oils on the touch screen to successful recover the unlock code. The smudges left on a screen when entering a password pattern on an Android smartphone can be analysed using photographs taken under a variety of lighting and camera positions. In many situations full or partial pattern recovery is possible. This method is ideal as it gains access to the device, without changing code or installing applications. If adb is turned off the smudge attack should be attempted before attempting a JTAG dump.

Figure 10: Smudge Attacks

Viber and Phone call exploits

A lock screen vulnerability has been found for Android users who have the Viber app installed. Viber allows users from every major mobile platform to make free calls, texts, and share photos for free but it's the Android version that's causing issues. Bkav Internet Security found a way to bypass the Android lock screen by simply sending two messages to a victim's handset. The exploit takes advantage of Viber's pop up messages, which wakes the screen of the victim's phone. A message alert will pop up where the attacker can bring up the keyboard for a brief second. The final part of the exploit requires an attacker to send a second message and hit the "Back" key on the device, which unlocks the device, allowing full access to the phone's contents.

Viber is aware of the issue and plans to issue an update to rectify it, but in the mean time, Viber recommends users disable Pop-up notifications if they want to protect themselves from this exploit. To do this, open the Viber application. In settings there is a setting “New message popup”. Untick this and popup will be disabled. See figure 11.

While this exploit shows an easy way to circumvent the Android lockscreen, the reality is that this exploit requires two things that most attackers don't have. First, they'd need physical access to your phone. Without your phone, it wouldn't matter if it was locked or unlocked since the attacker couldn't do anything. Second, an attacker would need to have your Viber user information to send you a message. Even if your phone was stolen and the attacker somehow knew that you were a Viber user, they'd still need to send your specific phone a message. These two factors greatly limit the potential pool of attackers, not to mention the fact that there are millions of Android users and only some use Viber. Like most of these exploits it poses little threat to most users, but may prove an easy method for a forensic investigator to gain access to a device if needed. Vaas L. (April 2013)

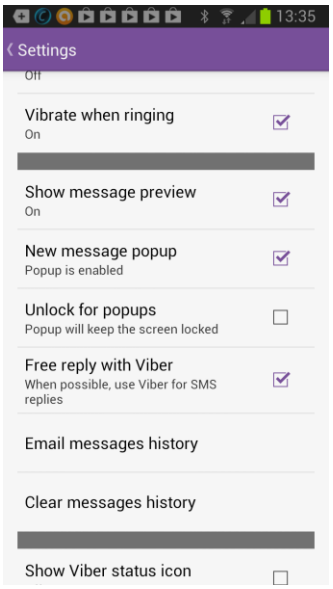


Figure 11: Disable Viber popups to avoid screen unlock exploit

A Samsung security vulnerability discovered by Blogger Terence Eden shows a way to briefly bypass lock screen security on Samsung Galaxy Note 2, momentarily allowing access to the home screen. By hitting "emergency call," then "emergency contacts," then holding the home button, the main home screen becomes visible for around a second -- just enough time to load an app, before reverting back to the lock screen. This dismisses any app that's loaded, but if a direct dial shortcut is placed on the home screen then it's possible to activate this and make a call, bypassing the lock screen security. Using this method it could also be possible to load up email or SMS apps for long enough to get an overview of sensitive messages.

With precise timing and a bit of patience, it's possible to use these windows of usability to load Google Play, use voice search to find a screen unlocker app and run it, thus removing the lock screen security. So in order to use this time is needed with the phone, the ability to use voice search inconspicuously and the patience to correctly hit the required sequence of screen taps. The point forensic investigators should take from this section, is that clever and simple screen unlock methods exist and they should be attempted before taking the route of the time consuming and complicated JTAG procedure. Dent S. (March 2013)

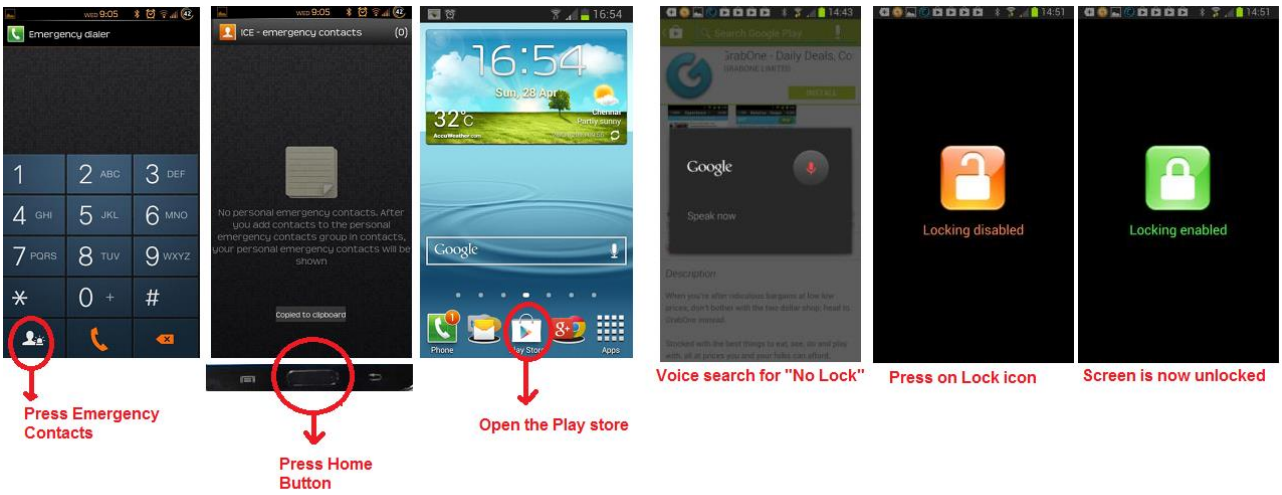


Figure 12: Samsung Galaxy Note 2 Screen Unlock Exploit

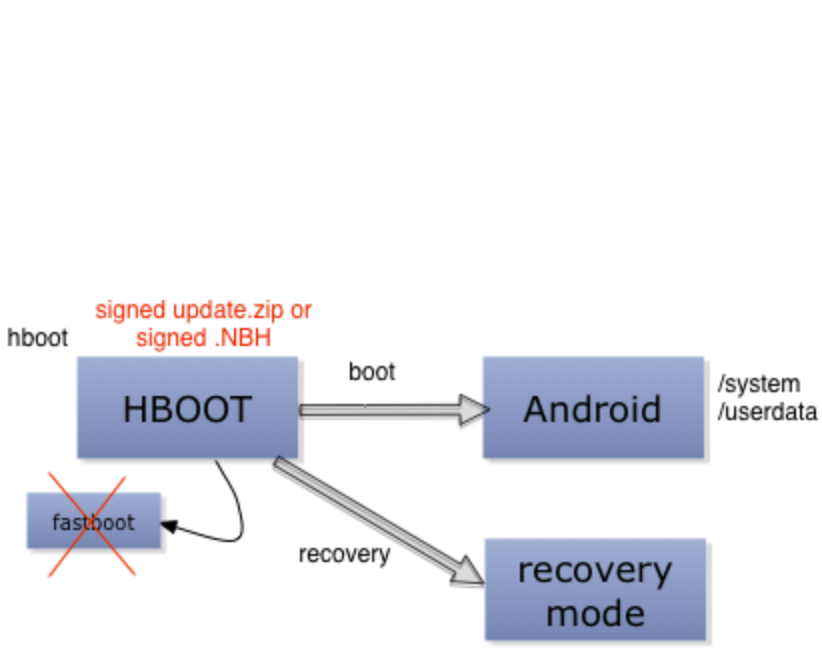
3.3 Unlocking the Bootloader

An unlocked bootloader is needed to install custom Firmware on an Android device. Not all devices ship with a locked bootloader, but if during the investigation it is discovered that the bootloader is locked the following procedure must be carried out. If the bootloader of the device is already unlocked, skip to the rooting procedure. In this research case, the phone chosen for examination was the HTC Wildfire S, which came with a locked bootloader. It required the bootloader unlocked, in order to proceed with rooting the device.

The default boot process takes place by starting code from bootloaders. HBOOT is a bootloader that is stored in NAND's first partition, mtd0 (if partition map is MTD, which are generally found on NAND devices with YAFFS2 filesystem). It is loaded in memory (RAM) when device is switched ON. Its jobs are:

- Check the Hardware.
- Initialize the Hardware.
- Start the Operating System (Either Android or Recovery).

HBOOT can also be used for flashing customs ROMs. The Android operating system can then be booted up in recovery mode. HBOOT allows communication with a freeware program called Fastboot.



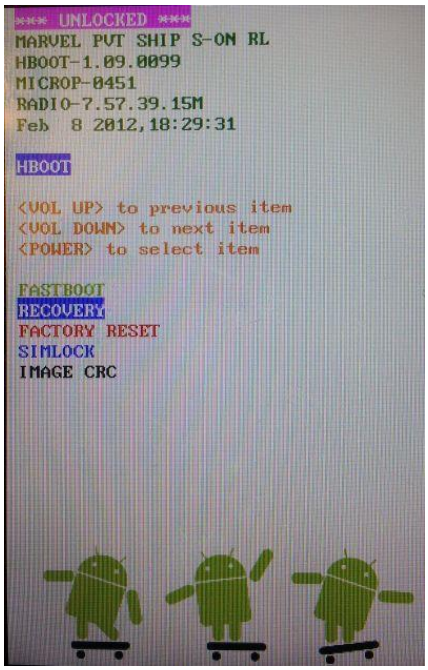


Figure 13: The HTC Wildfire S Bootloader

Sogeti ESEC Lab (June 2012)

By default, HBOOT allows reflashing of the boot or recovery partition by putting a signed update.zip file or by putting a signed .nbh file on the sdcard. Everything needs to be signed, or it will not be permitted. The fastboot protocol that is proposed by the AOSP (Android Open Source Project) does not allow reflashing of the device by default. The stock recovery partition allows resetting of the device to its original state, whilst erasing the data partition.

Unlocking a device allows the use of "fastboot" to reflash the device. Once unlocked it is possible to flash any custom unsigned ROM. This process needs to erase the user partition so it cannot be used in a forensics analysis.

S-OFF

By putting the device in S-OFF the security flags of a device are disabled and the restrictions are removed from HBOOT. If the device is S-OFF, HBOOT will accept unsigned custom ROM, update.zip or .nbh file. It is intended for manufacturers engineering phones to allow developers to reflash all partitions, but hackers and forensic examiners can use this method to flash custom images. (Sogeti ESEC Lab June 2012)

Unlocking HTC Wildfire S Bootloader

Remove and reinsert the battery then proceed to step 2. For devices without a removable battery, long press the power key then select restart. Hold down the volume down key while restarting to start the device in Bootloader mode.

Step 2

Press Volume Down and Power to start the device into Bootloader mode.

Step 3

Use the Volume buttons to select up or down. Highlight Fastboot and press the Power button.

Step 4

Connect the device to the computer via a usb cable

Step 5

From Ubuntu computer (windows will also work) you will need to download and install the Android Software Development kit (SDK), with ADT (Android Developer Tools). Download fastboot and copy it to /android-sdk-linux/sdk/platform-tools and navigate to the platform-tools folder in terminal.
Enter commands from Linux terminal:

```
root@darragh:/android-sdk-linux/sdk/tools# cp fastboot /bin/
 root@darragh:/android-sdk-linux/sdk/tools# cp adb /bin/
```

This will allow user to call fastboot and adb from anywhere in the terminal.

Now update the sdk to support the API of the phone.

To launch SDK Manager; type;

```
root@darragh:/android-sdk-linux/sdk/tools#  ./android.
```

Update the relevant API packages for the device

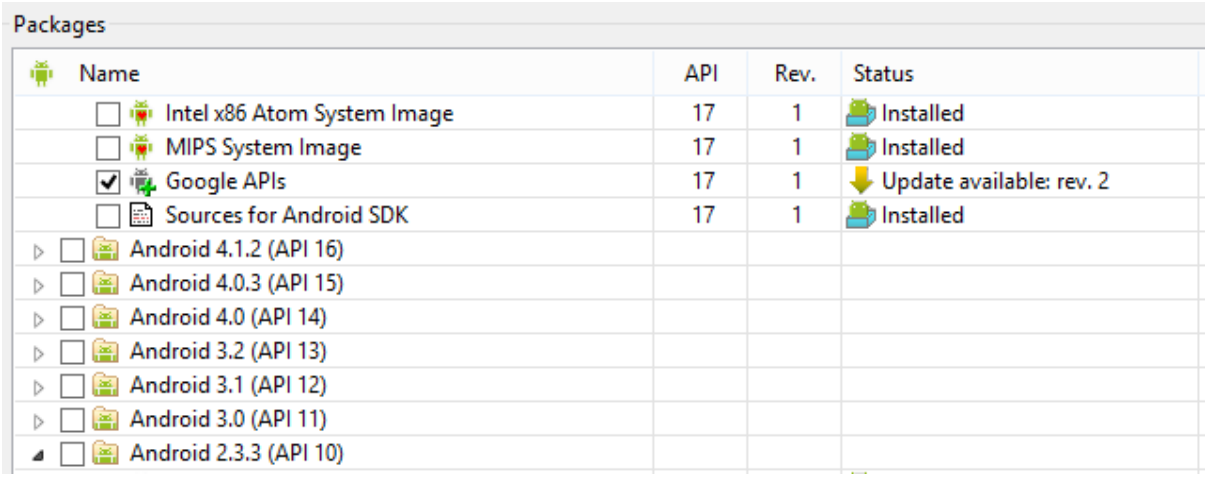


Figure 14: The SDK Manager, API Updates

Now retrieve the Identifier token from the phone;

```
root@darragh:/android-sdk-linux/sdk/tools#   fastboot oem get_identifier_token
```

Record the token and submit it online to http://www.htcdev.com

A reply email with the attachment: **Unlock_code.bin** will be returned. Save this file in the same folder as your fastboot files

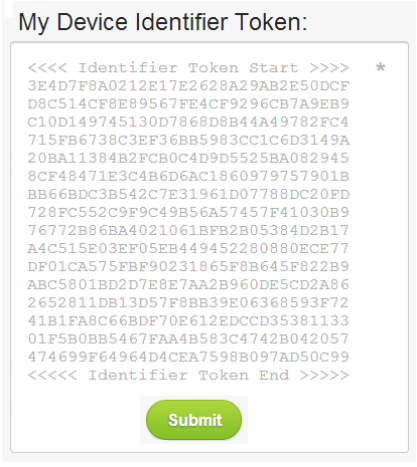


Figure 15: HTC Bootloader Unlock Token

In the terminal CLI type:

```
root@darragh:/android-sdk-linux/sdk/tools# fastboot flash unlocktoken Unlock_code.bin
```

In the terminal the following message is displayed:

```
sending 'unlocktoken' (0 KB)...
```

```
OKAY [  0.330s]
writing 'unlocktoken'...
(bootloader) unlock token check successfully
OKAY [  0.007s]
finished. total time: 0.338s
```

On the phone a disclaimer is displayed. Please read this carefully as this action may void the warranty. Use the Volume buttons to highlight a choice, and the Power button to make the selection. If Yes is selected, the phone will be reset to its' factory default settings, and the bootloader will be unlocked. If No is selected, your phone will reboot and no modifications will have been made. (HtcDev n.d.);

Figure 16: Bootloader Unlock Application

This method is a free method of unlocking the phone and was sufficient for unlocking the bootloader for research and development purposes, but because it restores it to factory settings and wipes user data, it could not be used in a forensic investigation.

Forensics method - Disabling security

If the bootloader needs to be unlocked for a forensic examination, the XTC Clip is a standalone clip that allows you to unlock HTC Android phones without resetting the device to factory settings. The hardware can be purchased online. It will disable the security flag of HTC phones, while not erasing the user data. It supports (almost) all HTC phones and HBOOT versions. This tool uses a signed debug .NBH file and a "fake" SIM card to request the baseband to make the device S-OFF.

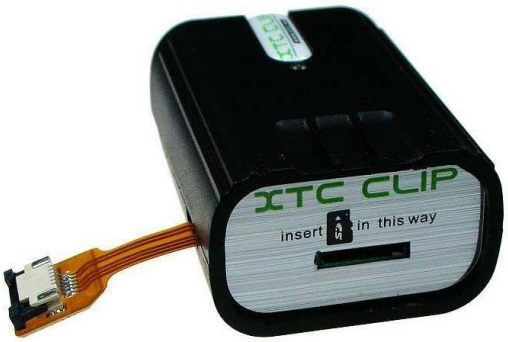


Figure 17: The XTC Clip

Techgsm (n.d.)

Loading an alternative boot

Once security is disabled, it is possible to push an "update.zip" file containing a custom recovery that allows booting in recovery mode and extract user data. After the forensic examination it is possible to restore the original recovery with the exact same procedure. Another method consists of pushing an "update.nbh" file on the sdcard with code to disable the LOCK state. Then, reboot in HBOOT mode and use the fasboot commands. It is less intrusive and also reversible.

Restoration

To restore the device to its original state, it is possible to reflash an original recovery image, relock the device using a custom "update.nbh" file. Restoring the security flag is done by directly sending a command to the baseband. This procedure takes place in 4 steps:

- Put the device in HBOOT mode (hold down and press power).
- Select the fastboot mode from the menu.
- Issue the "fastboot oem rtask C" command from the PC
- Send the "AT@SIMLOCK=7,1" command to one of the USB serial.

Countermeasures

This procedure is used to access the personal data or inject a backdoor in the mobile phone. The XTC clip was to disable the security. Disabling the XTC clip capabilities is possible by replacing the RSA public key of HTC in HBOOT. This procedure needs to be made manually and it may void the warranty but it is the only known solution to this problem. Consequently, the debug "update.nbh" files (signed by HTC) will not be accepted anymore by the device. (Sogeti ESEC Lab June 2012)

3.4 Rooting Techniques

Root privileges on Android devices are not enabled by default. However, it is possible to gain root privileges in certain scenarios, several of which we will cover next. There are some major challenges to obtaining root privileges to keep in mind though:

- Gaining root privileges changes the device in many situations.
- The techniques for root privileges differ not only for each manufacturer and device but for each version of Android and even the Linux kernel in use. Just based on the Android devices and versions developed to date, there are literally thousands of possible permutations.
- Many of the exploits used to gain root privileges are discussed online and often contain inaccurate information.

Given this, gaining root privileges can be quite difficult and frustrating. There are three primary types of root privileges:

- Temporary root privileges attained by a root exploit which does not survive a reboot. Typically the adb daemon is not running as root in this instance.
- Full root access attained through a custom ROM or persistent root exploit. Custom ROMs often run the adb daemon as root while most of the persistent root exploits do not.
- Recovery mode root attained by flashing a custom recovery partition or part of a custom ROM. Custom ROMs often run the adb daemon as root as do most of the modified recovery partitions.

Android enthusiasts who want root access are typically only interested in full, sustained root privileges. However, from a forensics standpoint, temporary root privileges or root access via a custom recovery mode are preferred. A Universal rooting approach has been created by XDA developer Adam Outler, who has written his own scripting language for deploying exploits over the Android Debug Bridge (ADB). The Cross-platform ADB Scripting Unified Android Loader (CASUAL) is a project focused on connecting to Android devices and feeding it exploits over ADB, which often provides a much faster root method. In theory, CASUAL doesn't require a screen, mouse, or even a keyboard to operate. He has built a silver box housing a Raspberry Pi computer with an Arduino battery backpack. The box has five LEDs and a toggle switch on top of the box, and these are used to communicate with the user. When a phone is connected to the Raspberry Pi, the LEDs will signal connection status of the phone and running the exploits. Finally a pass or fail LED will illuminate to display if rooting has passed/failed. The success rate of the box is approximately 70%.
The exploits loaded on the box handle nearly all standard Android firmware versions, and due to the open nature of CASUAL anyone can contribute when they have found new exploits. The box is something anyone can build for themselves, using the Headless mode designed for embedded devices like the Raspberry Pi.

Rooting the HTC Wildfire S with recovery mode root

Now that the bootloader on the HTC Wildfire S is unlocked, the phone can be rooted. Unzip the Wildfire_S_CWM_Recovery archive, downloadable from the XDA Developers website. In the extracted contents there is a recovery.img file. Start the device into bootloader mode and select fastboot.

Once fastboot is enabled enter the command below.

```
root@darragh /platform-tools# fastboot flash recovery recovery.img
```

Reboot the phone and connect USB. When prompted to connect a USB connection type, select; Mount a Disk Drive. Now copy the root.zip (also from XDA Developers) to the root of the sdcard.

Power down the phone and start the device into bootloader mode. Select Recovery. The Clock Work Mod recovery will begin. Select install zip from sdcard.

->Next screen, Select: install zip from sdcard
->Next screen, Select: root.zip
->Next screen, Select: install root.zip
The upgrade is complete

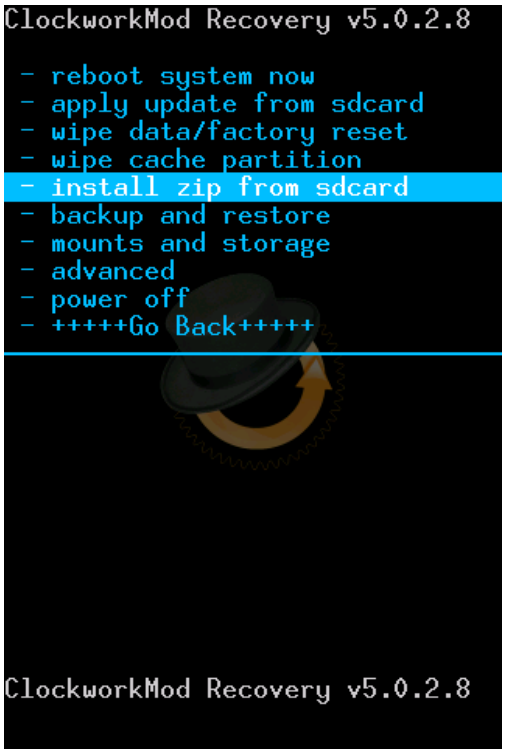


Figure 18: Rooting HTC Wildfire S with ClockworkMod Recovery

Superuser and Busybox are included in the root.zip file busybox is in /system/xbin. This will provide you with Nanddump and dd to take partition images.

```
/home/sdk/platform-tools# adb shell
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
Android$ su
Android#
```

(The phone is now rooted)

Rooting the Samsung Galaxy Note 2

To root the Samsung Galaxy Note 2 a one-click Rooting method can be achieved with the Framaroot application. For this research study, the Samsung Galaxy Note 2 was successfully rooted using this method. On December 15, 2012, a member of the XDA Developer Forums going by the name "Alephzain" published a vulnerability affecting all Android devices using the Samsung Exynos chipset and running Android 4.0 (Ice Cream Sandwich) or greater. Affected devices include the Samsung Galaxy S3 (the North American version is not affected because it uses a Qualcomm chipset instead) and Exynos variants of the Galaxy S2, Galaxy Note, Galaxy Note 2, and Galaxy Tab.

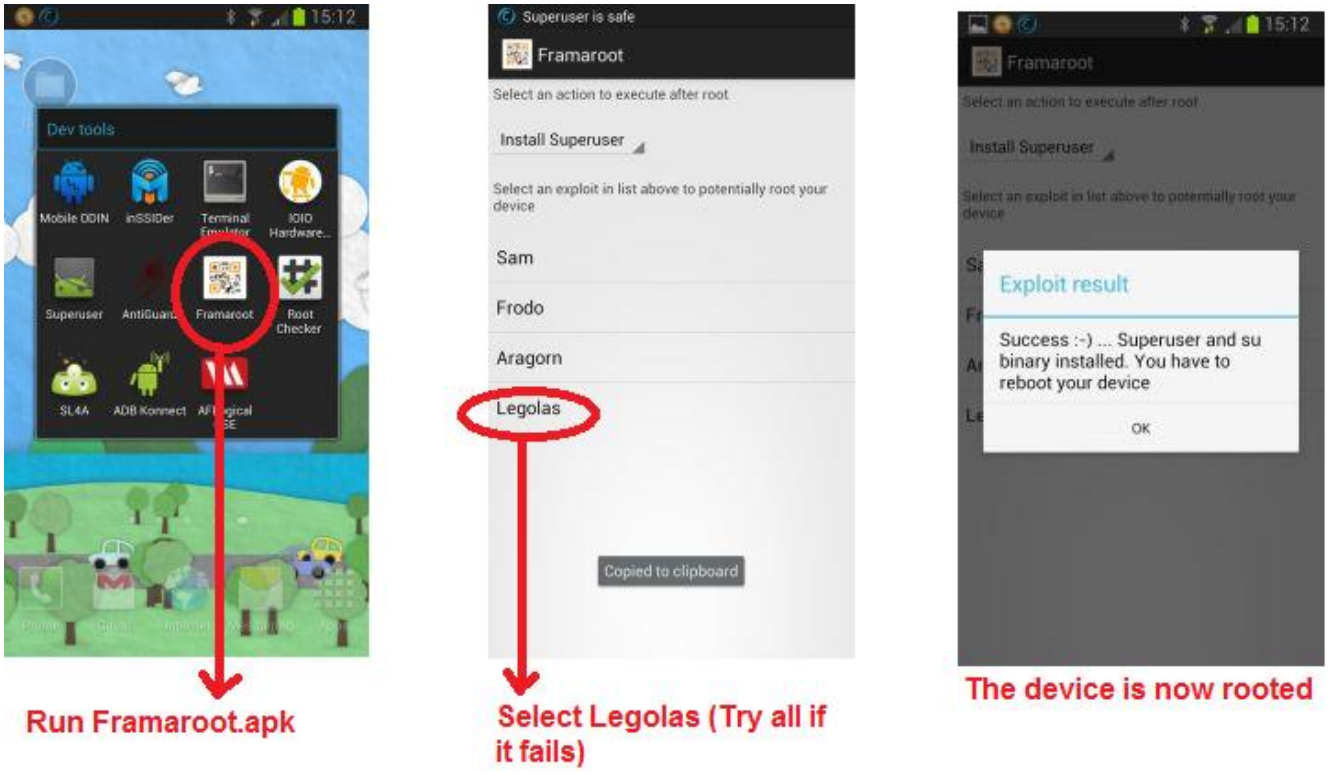


Figure 19: Rooting Samsung Galaxy Note 2 with Framaroot

How Framaroot Works

The vulnerability allows any unprivileged user to read and write to arbitrary physical memory on an affected device by mmap()-ing a file descriptor to the world-writable device file at */dev/exynos-mem*. Alephzain's exploit utilizes this capability by mapping the kernel address space, modifying a format string used by the kptr_restrict security feature in order to disable it, and finally modifying the .text segment of the kernel itself in order to trigger a privilege escalation payload and gain root privileges. The original exploit is available on XDA. Shortly after the publication of this vulnerability, Samsung released updates for several of its devices, including the Galaxy S3.

On February 2, 2013, Alephzain published an APK version of his exploit, which he

called Framaroot. Examining this APK revealed that Alephzain had added exploits for additional vulnerabilities besides the original Exynos flaw. (Alephzain July 2013)

Analyzing Framaroot

To reverse engineer Framaroot, decompile the Framaroot APK to Java using dex2jar and jad. The *alephzain.framaroot.FramaActivity* revealed the following logic:

```
public native String[] Check();
public native long Launch(String s);
protected void onCreate(Bundle bundle)
{
String as[] = Check();
if(as.length == 0)
 {
/* Device not affected, exit */
...
 } else
      {
...
/* Device affected, launch new thread to exploit */
LaunchThread launchthread =
new LaunchThread(adapterview.getItemAtPosition(i).toString());
 launchthread.start();
  ...
     }
 }
```

The above code first invokes the **Check()** native method to probe for the existence of a supported vulnerability, and if successful, launches a new thread that invokes the native **Launch()** method. These methods are both implemented in a bundled dynamic library, *lib/armeabi/libframalib.so*. Reverse engineering the **Check()** method in the bundled library revealed that Alephzain had defined a structure containing information for each supported target.

This structure can be seen below;

```
struct target {
char *tag;
char *device_name;
 int fd;
 int flags;
 unsigned long offset;
 unsigned long size;
```

```
unsigned long start_offset;
 unsigned long device_len;
int (*)(void) func1;
 int (*)(void) func2;
};
```

Possible case once application is launched

- A popup saying "Your device seems not vulnerable to exploit included in Framaroot", in this case you can uninstall app
- You seeing one or more exploit name, also click on one after you have selected an action and you will see one of the above messages

Possible case once exploit is launched

- "Success - Superuser and su binary installed. You have to reboot your device"
- "Failed - Exploit work but installation of Superuser and su binary have failed"
- "Half-Success - system partition is read-only, use local.prop trick. Reboot your device and use adb to see if it run as root", happen when the filesystem in use on system partition is a read only filesystem (ex: squashfs)
- "Failed - Try another exploit if available" (Rosenberg D. February 2013)

3.5 Software-Based Logical Extraction Techniques

Logical acquisition extracts a bit-by-bit copy of logical storage objects (e.g., directories and files) that reside on a logical store (e.g., a file system partition). Logical acquisition has the advantage that system data structures are easier for a tool to extract and organize. Logical extraction acquires information from the device using the original equipment manufacturer application programming interface for synchronizing the phone's contents with a personal computer. If a phone is rooted, more information can be gained from otherwise protected data.

A logical extraction is generally easier to work with as it does not produce a large extracted image file. However, a skilled forensic examiner will be able to extract far more information from a physical extraction. Logical extraction usually does not produce any deleted information, due to it normally being removed from the phone's file system. However, in some cases, particularly with platforms built on SQLite, such as iOS and Android, the phone may keep a database file of information which does not overwrite the information but simply marks it as deleted and available for later overwriting. In such cases, if the device allows file system access through its synchronization interface, it is possible to recover deleted information. File system extraction is useful for understanding the file structure, web browsing history, or app usage, as well as providing the examiner with the ability to perform an analysis with traditional computer forensic tools._Software-based physical techniques have a number of advantages over the hardware based techniques. Software-based techniques:

- Are easier to execute.
- Often provide direct access to file systems to allow a complete copy of all logical files (simplifies some analysis).
- Provide very little risk of damaging the device or data loss. (Hoog 2011)

Examples of logical extraction methods are;

1. ADB Pull

The adb pull can be used to extract files and directories from the Android device. Unless it has root access or is running a custom ROM, the adb daemon can only be run with shell permissions; therefore some of the more forensically relevant files are not accessible. However, there are still files which can be accessed.

- On non-rooted devices, an adb pull can still access useful files such as unencrypted apps, most of the tmpfs file systems that can include user data such as browser history, and system information found in "/proc," "/sys," and other readable directories.
- On rooted devices, a pull of nearly all directories is quite simple and certain files and directories from "/data" would be of interest.

With Android SDK installed and the phone connected to the computer, to perform logical extraction use adb to copy files from the phone to the computer. This is done with the "adb pull" command. Here are the commands:

```
dmerrick@ubuntu:~$ cd <android-sdk-dir>/platform-tools
dmerrick@ubuntu:~$./adb pull
/data/data/com.android.providers.telephony/databases/mmssms.db
dmerrick@ubuntu:~$./adb pull
/data/data/com.android.providers.contacts/databases/contacts2.db
```

After executing this commands the databases have been extracted to the "<android-sdk-dir>/platform-tools" folder. To examine the databases use an SQLite browser, such as the SQLite Manager addon for Firefox or SQLite Database Browser for Ubuntu. To alter files such as settings.db to remove lock pattern or pattern lock out, this is set when the incorrect pattern has been entered too many times;

```
sqlite3 settings.db
update system set value=0 where name='lock_pattern_autolock';
update system set value=0 where name='lockscreen.lockedoutpermanently';
```

To put them back on the phone, remount the /system partition in read-write mode, delete the present database files (if there are present) and push the files back on the phone. First check to see which physical partitions are mounted on "/system" to know where you should remount it:

./adb shell mount

Check which physical partition is mounted at "/system" and remount that partition as read-write, delete the databases and push the altered databases back in their place;

```
ubuntu:~$ ./adb shell mount -o remount,rw -t yaffs2 /dev/block/mtdblock3 /system
ubuntu:~$./adb shell rm /data/data/com.android.providers.telephony/databases /mmssms.db
ubuntu:~$./adb shell rm /data/data/com.android.providers.contacts/databases/ contacts2.db
ubuntu:~$./adb push mmssms.db /data/data/com.android.providers.telephony/databases/
ubuntu:~$./adb push contacts2.db /data/data/com.android.providers.contacts/databases/
```

When utilizing the physical technique, it is not always possible to mount some acquired file systems such as YAFFS2. If adbd is running with root permissions, it is possible to quickly extract a logical copy of the file system with adb pull. As adb is not only a free utility

in the Android SDK but also very versatile, it should be, one of the primary logical tools used on a device.

(2) AFLogical
AFLogical is an Android forensics logical technique which is distributed free to law enforcement and government agencies. The app, developed by viaForensics, extracts data using Content Providers, which are a key feature of the Android platform. This is the same technique that commercial forensics tools use for logical forensics.
Here is a quick recap of the key components of Android's security model:

- Each application is assigned a unique Linux user and group id.
- Apps execute using their specific user ID in a dedicated process and Dalvik VM.
- Each app has dedicated storage, generally in "/data/data," that only the app can access.

However, the Android framework does provide a mechanism by which apps can share data. An app developer can include support for Content Providers within their application, which allows them to share data with other apps. The developer controls what data is exposed to other apps. During the install of an app, the user controls whether or not an app should gain access to the requested Content Providers. Some examples of Content Providers are:
_ SMS/MMS
_ Contacts
_ Calendar
_ Facebook
_ Gmail

The AFLogical app takes advantage of the Content Provider architecture to gain access to data stored on the device. Similar to commercial Android logical tools, USB debugging must be enabled on the device for AFLogical to extract the data. AFLogical extracts data from the Content Providers and provides the output information to the SD card in CSV format and as an info.xml file, which provides details about the device and installed apps. AFLogical supports devices running Android 1.5 and later, and has been specifically updated to support extraction of large data sets such as an SMS database with over 35,000 messages.

The currently supported Content Providers are:
1. Browser Bookmarks
2. Browser Searches
3. Calendars
4. Calendar Attendees
5. Calendar Events
6. Calendar Extended Properties
7. Calendar Reminders
8. Call Log Calls
9. Contacts Contact Methods
10. Contacts Extensions
11. Contacts Groups
12. Contacts Organizations
13. Contacts Phones
Logical techniques 221
14. Contacts Settings
15. External Media
16. External Image Media
17. External Image Thumb Media
18. External Videos
19. IM Account

20. IM Accounts
21. IM Chats
22. IM Contacts Provider (IM Contacts)
23. IM Invitations
24. IM Messages
25. IM Providers
26. IM Provider Settings
27. Internal Image Media
28. Internal Image Thumb Media
29. Internal Videos
30. Maps-Friends
31. Maps-Friends extra
32. Maps-Friends contacts
33. MMS
34. Mms Parts Provider (MMSParts)
35. Notes
36. People
37. People Deleted
38. Phone Storage (HTC Incredible)
39. Search History
40. SMS
41. Social Contracts Activities

AFLogical OSE (Open Source Edition) will only capture, CallLog Calls, Contact Phones, MMS, MMSParts and SMS. For full functionality of AFLogical, it would need to be purchased from ViaForensics

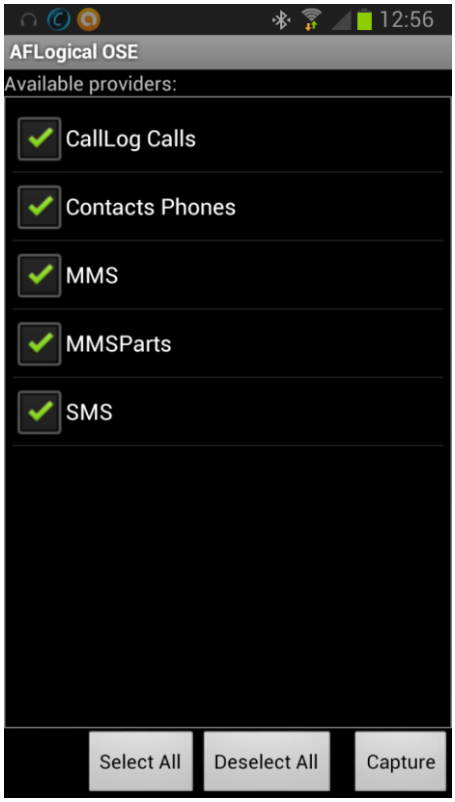


Figure 20: ViaForensics AFLogical OSE

3.6 Software-Based Physical Extraction Techniques

Physical acquisition implies a bit-for-bit copy of an entire physical store (e.g. flash memory); therefore, it is the method most similar to the examination of a personal computer. A physical acquisition has the advantage of allowing deleted files and data remnants to be examined. Physical extraction acquires information from the device by direct access to the flash memories. Generally this is harder to achieve because the device original equipment manufacturer needs to secure against arbitrary reading of memory; therefore, a device may be locked to a certain operator. To get around this security, mobile forensics tool vendors often develop their own boot loaders, enabling the forensic tool to access the memory (and often, also to bypass user passcodes or pattern locks).

Software based physical acquisition techniques run as software on a device with root access and provide a full physical image of the data partitions. Generally the physical extraction is split into two steps, the dumping phase and the decoding phase. To execute the software-based physical techniques, you first must gain root privileges and then run the acquisition programs. Examples of software-based physical techniques are:

1. Fastboot

Fastboot is another utility that flashes images to the NAND flash over USB. The source code for fastboot is contained in the AOSP and thus, the utility is built when compiled with the AOSP code. Like sbf_flash, the boot loader must support fastboot, which not only requires a compatible boot loader but also one that has security turned off (S-OFF).To enter fastboot mode, power up the device (or reboot it) while holding down the BACK key. Hold the BACK key down until the boot loader screen is visible and displays “FASTBOOT.” The device is now in fastboot mode and is ready to receive fastboot commands. For ADP devices, the boot loader screen shows an image of skateboarding robots. Other devices may show a different image or colour pattern. In all cases, the boot loader screen shows the text “FASTBOOT” when in fastboot mode. The boot loader also shows the radio version.

2. Sbf_flash

Similar to Motorola’s RSD Lite is a utility called sbf_flash that does not carry the license and usage restrictions of RSD Lite. The application was developed and posted online by an Android enthusiast. This utility was developed on Linux, and now also runs on OS X, and thus, greatly simplifies the flashing of data to the NAND flash via an unlocked boot loader. The sbf_flash utility looks for a device in bootloader mode and immediately flashes the image file to the NAND flash. The status of the update process is displayed on screen and afterwards the Droid is rebooted.

3. AFPhysical Technique:

The AFPhysical technique was developed by viaForensics to provide a physical disk image of Android NAND flash partitions. The technique requires root privileges on the device and should support any Android device. The technique, however, is not a simple process and the forensic analyst will have to adapt the technique for the specific device investigated. This is a direct result of the large variations in Android devices not only between manufacturers but devices running different versions of Android. The overall process for AFPhysical is quite simple:

- Acquire root privileges on the target Android device.
- 2. Identify NAND flash partitions which need to be imaged.
- 3. Upload forensic binaries to the target Android device.
- 4. Acquire physical image of NAND flash partitions.
- 5. Remove forensic binaries if any were stored on non-volatile storage.

Hoog (2011)

AFPhysical Procedure

Step 1: Creating ARM Forensic Tools

The Android Linux kernel does not include tools that will be needed. The rooting method used previously mounting recovery image, may have included busy box. The tools needed are;

1. Nanddump – Dumps the Nand partition
2. Netcat – Will forward the Nanddump output to the Forensic Workstation, to avoid writing to the smartphone
3. Md5sum – will calculate cryptographic hash, which is a digital fingerprint of the partition to verify it was unchanged during the transfer
4. Tar – To extract Nanddump, NC and Md5sum from the archive once it is pushed onto the target device.
5. dd – dumps the EXT4 and FAT16/32 partitions

The best forensics approach is to locate each of the tools and cross compile them from source code for the ARM processor. If pre-compiled tools can be found online, they can be used, but is advisable for any Android forensics expert to build an ARM toolchain into linux, so that any forensics tools needed can be cross compiled and used, without relying on others to provide the cross-compiled tools. Buildroot is a set of Makefiles and patches that makes it easy to generate a complete embedded Linux system. Buildroot can generate any or all of a cross-compilation toolchain, a root filesystem, a kernel image and a bootloader image. It can be used to cross-compile tools for ARM devices.

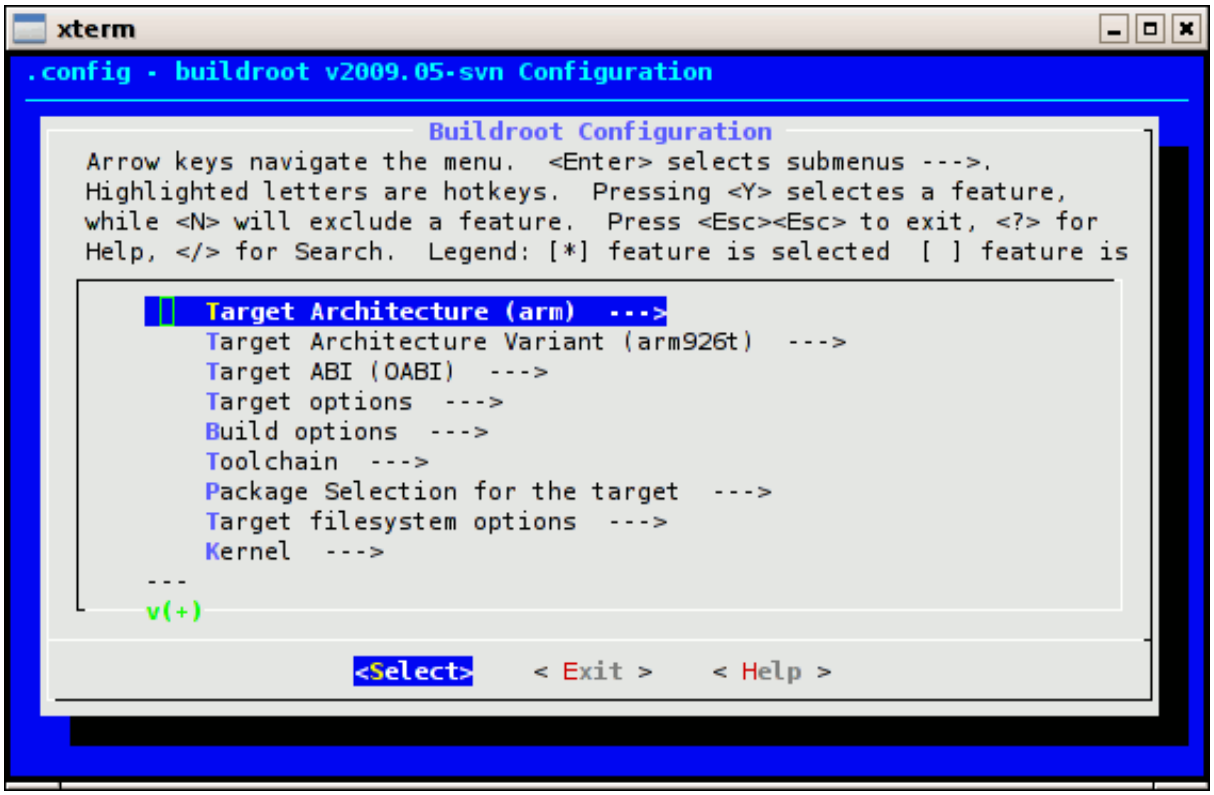


Figure 21: Cross-compiling with Buildroot

Step 2: Installing Forensic tools on Android device

Once the tools have been cross-compiled, compress them into a single archive called Android_Physical.tar using tar. To avoid writing any data to the NAND flash, examine the output of the mount command and take note that the "/dev" directory is tmpfs which is stored in RAM.

```
/ # mount
rootfs on / type rootfs (rw)
tmpfs on /dev type tmpfs (rw,mode=755)
devpts on /dev/pts type devpts (rw,mode=600)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
/dev/block/mtdblock7 on /cache type yaffs2 (rw,nodev,noatime,nodiratime)
```

It is best to push the forensic utilities to "/dev", to avoid writing to the suspect device and altering the user data. This action should be recorded in the examiners journal, stating command executed, files pushed, where they were pushed and why. Next make the programs executable on the device. To achieve this, use the chmod command, which changes the execute flag in the file permissions.

```
dmerrick@ubuntu:~$ adb shell
/ # cd /dev/ Yaffs2_Physical.tar
/dev/ Yaffs2_Physical.tar # ls –l
        -rw-rw-rw- 1 0 0 711168 Jan 24 2011 md5sum
        -rw-rw-rw- 1 0 0 669799 Jan 24 2011 nanddump
        -rw-rw-rw- 1 0 0 711168 Jan 24 2011 nc
        -rw-rw-rw- 1 0 0 711168 Jan 24 2011 tar
        -rw-rw-rw- 1 0 0 711168 Jan 24 2011 dd
/dev/AFPhyiscal # chmod 755 *
/dev/AFPhyiscal # ls –l
        -rwxr-xr-x 1 0 0 711168 Jan 24 2011 md5sum
        -rwxr-xr-x 1 0 0 669799 Jan 24 2011 nanddump
        -rwxr-xr-x 1 0 0 711168 Jan 24 2011 nc
        -rwxr-xr-x 1 0 0 711168 Jan 24 2011 tar
```

Once the "chmod 755" command has been executed on the programs, they each have the execute bit now set, which is represented by the "x" in the file permissions.

```
dmerrick@ubuntu:~$ adb push Yaffs2_Physical.tar / /dev/ Android_Physical
        push: Yaffs2_Physical.tar /tar -> /dev/Android_Physical.tar /tar
        push: Yaffs2_Physical.tar /md5sum -> /dev/ Android_Physical.tar l/md5sum
        push Yaffs2_Physical.tar /nanddump -> /dev/ Android_Physical.tar /nanddump
        push: Yaffs2_Physical.tar l/nc -> /dev/ Android_Physical.tar /nc
        push: Yaffs2_Physical.tar l/nc -> /dev/ Android_Physical.tar /dd
        4 files pushed. 0 files skipped.
        1003 KB/s (2803303 bytes in 2.727s)
```

Step 3: Creating images of mounted partitions on Android device

There are 4 Android physical acquisition strategies that can be used on a device with root access:

1. Full nanddump of all partitions on YAFFS2 filesystem, including data and OOB (preferred).

2. A dd image of partitions on YAFFS2 filesystem, which only acquires the data, not the OOB.
3. A full dd of all partitions on EXT4 filesystem, which will copy all filesystem data.
4. A full dd of the External Sdcard's FAT filesystem, which will copy all filesystem data.

In addition, there are two primary ways to save the acquired data from the device:

1. Use adb port forward to create a network between the Ubuntu workstation and Android device over USB.
2. Place an SD card into the device, mount, and save locally.

There are advantages to both approaches. With adb port forwarding an sdcard is not inserted and files are copied directly onto the workstation. This transfer method is slower, but is does not write to the devices local storage. With an SD card inserted and data written directly to it, the acquisition is much faster, but it more invasive from a forensics point of view, than the previous method.

```
dmerrick@ubuntu:~$ adb shell
shell@Android/ # /dev/Android_Phyiscal/nanddump /dev/mtd/mtd5ro | /dev/ Android_Phyiscal/nc -l -p
12345
        ECC failed: 0
        ECC corrected: 0
        Number of bad blocks: 0
        Number of bbt blocks: 0
        Block size 131072, page size 2048, OOB size 64
        Dumping data starting at 0x00000000 and ending at 0x105c0000...
```

Now that the Android device is sending the nanddump data over Netcat it needs to be received on the Ubuntu machine side:

```
dmerrick@ubuntu:~$ nc 127.0.0.54321 > mtd8.nanddump
dmerrick@ubuntu:~$ ls -lh mtd8.nanddump
        -rw-r--r-- 1 dmerrick dmerrick 270M 2011-02-26 20:58 mtd5.nanddump
```

MD5 hash

Although the user data partition was not mounted on the device during acquisition, the md5sum hash signature of "/dev/mtd/mtd5ro" will change even without any writes. This is due to the nature of NAND flash where the operating system and memory are in a nearly constant state of change from wear levelling, bad block management, and other mechanisms which occur despite the lack of changes to the user data. The best approach is to perform an md5sum of the resulting NAND flash file to ensure integrity from that point forward. Take md5sum before and after imaging of each partition, it should be the same, otherwise an error has occurred during transfer.

```
shell@Android/ # /dev/AFPhyiscal/nanddump /dev/mtd/mtd8ro > /sdcard/af-book-mtd8.nanddump
        ECC failed: 0
        ECC corrected: 0
        Number of bad blocks: 1
        Number of bbt blocks: 0
        Block size 131072, page size 2048, OOB size 64
        Dumping data starting at 0x00000000 and ending at 0x105c0000...
shell@Android / # ls -l /sdcard/af-book-mtd8.nanddump
-rwxrwxrwx 1 0 0 283041792 May 13 16:54:33 /sdcard/ mtd5.nanddump
```

Once partitions have been copied, they can be analysed with forensic tools on the workstation.

3.7 Bypassing the Android Passcode

If an Android device has a defined passcode in place and it cannot (at first) be bypassed with a smudge attack or Screen lock exploit, a forensically sound method is needed to gain access to the device. If adb is not enabled on the device, the passcode is needed to turn it on, in order to physically image the device. There are 3 kinds of passcode for Android devices:

- Scheme
- PIN code
- Password or Swipe lock.

Getting sensitive data

From the recovery mode, it is possible to mount the user partition, bypassing the Android passcode. This will allow access to databases such as: calendar.db, mmssms.db, facebook.db, browser.db, telephony.db, contact2.db, etc.

The following files are needed to override the passcode.

- /data/data/com.android.providers.settings/databases settings.db,
- /data/system/password.key
- /data/system/gesture.key

Code:

```
adb shell
cd /data/data/com.android.providers.settings/databases
sqlite3 settings.db
update system set value=0 where name='lock_pattern_autolock';
update system set value=0 where name='lockscreen.lockedoutpermanently';
quit
-AND/OR-
method 2:
Cat /data/system/gesture.key
Cat /data/system/password.key
Code:
adb shell rm /data/system/gesture.key
```

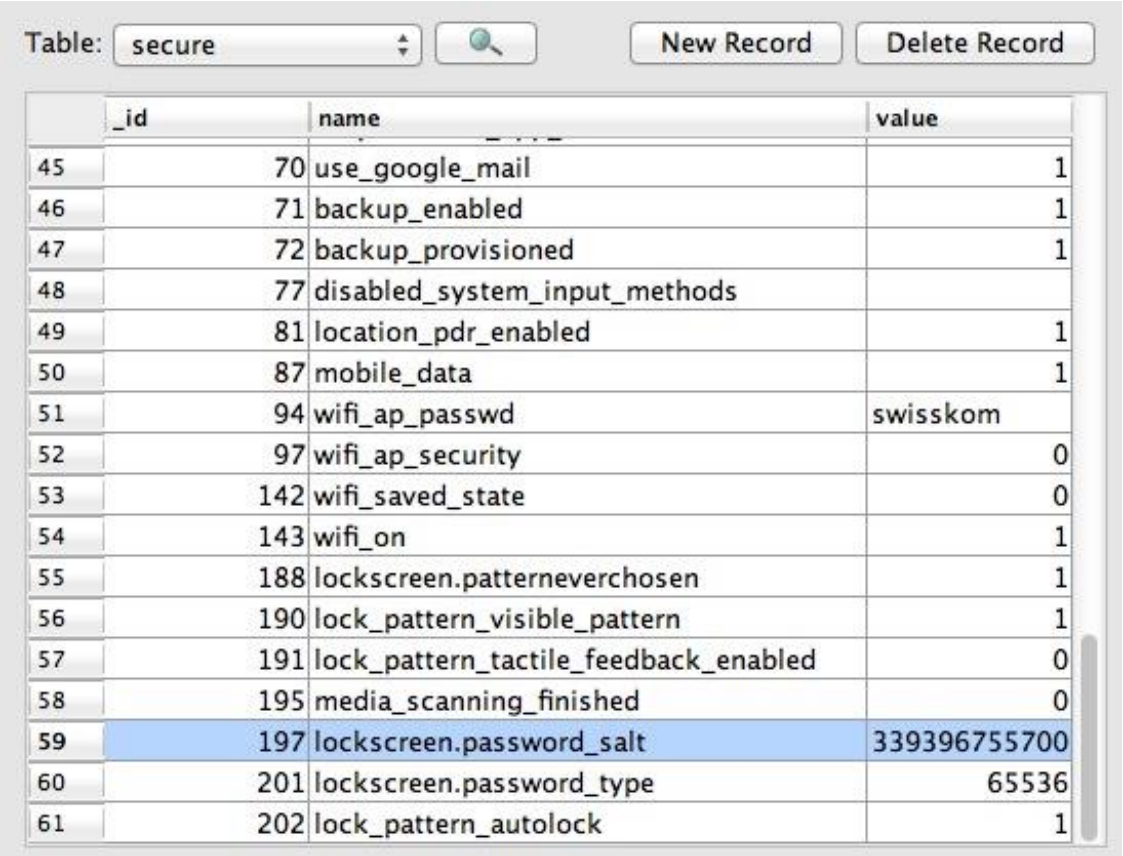
For cracking the password it is important to get the salt and allow enough time for attempting a brute force attack. The salt is a string of the hexadecimal representation of a random 64-bit integer. To get this salt, there are two ways depending on whether phone is rooted or not.

Obtaining the Salt on a Rooted Smartphone

If the smartphone is rooted and USB debugging is enabled, cracking of the pattern lock can be done quickly. Dump the file /data/system/password.key and the salt, which is stored in a SQLite database under the lockscreen.password_salt key. The corresponding database can be found in /data/data/com.android.providers.settings/databases and is called settings.db (see the figure below). Once both strings have been obtained, start brute forcing the password.(Forensic Blog February 2012)

Run the Android Brute Force Encryption cracking program from Santoku against password.key and lockscreen.password_salt. To run the program, launch it under Santoku -> Device Forensics -> Android Brute Force Encryption. Run the script;

```
Santoku# python bruteforce_stdcrypto.py /home/Forensics/ password.key /home/Forensics/
lockscreen.password_salt key
```



	_id	name	value
45	70	use_google_mail	1
46	71	backup_enabled	1
47	72	backup_provisioned	1
48	77	disabled_system_input_methods	
49	81	location_pdr_enabled	1
50	87	mobile_data	1
51	94	wifi_ap_passwd	swisskom
52	97	wifi_ap_security	0
53	142	wifi_saved_state	0
54	143	wifi_on	1
55	188	lockscreen.patterneverchosen	1
56	190	lock_pattern_visible_pattern	1
57	191	lock_pattern_tactile_feedback_enabled	0
58	195	media_scanning_finished	0
59	197	lockscreen.password_salt	339396755700
60	201	lockscreen.password_type	65536
61	202	lock_pattern_autolock	1

Figure 22: The Settings.db Database

Forensic Blog (February 2012)

Obtaining the Salt on a Non-Rooted Smartphone

If it is a non-rooted smartphone, a physical hardware based extraction is required. A Riff-Box and a JIG-adapter or some soldering skills are needed to JTAG the smartphone. Hardware methods involve connecting hardware to the device or physically extract device components. The hardware based methods required specialized and often expensive equipment and training but can be very effective on devices where root access is unattainable. The software based physical techniques are a more direct path to acquisition and are often the best place to start. Of course, before software based techniques are possible, you must have root access on the device.

This is an extraction method, which involves physically connecting hardware to the circuit board and taking a dump of the devices storage. Joint Test Action Group (JTAG) was originally developed for testing printed circuit boards and later became standardized as the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. It is a standard for test access ports on PCB using boundary scan. It provides an interface with commands to be used for diagnostics and debugging of the hardware components. Boundary scan refers to a test of the input and output pins of a component to determine proper functioning and correct interconnections between components. JTAG ports on a PCB can be used to access the flash memory using extest mode or debug mode.

For cell phones with PCB's that do support JTAG, each model may have different implementation of the standard specific to that phone. The Flash memory chips themselves are not JTAG enabled, however, they are connected to other components such as the processor which can be used to gain access to the flash memory. This is assuming that the processor is JTAG enabled. JTAG ports are usually located on the edge of the PCB in a row and therefore, may be easy to locate.
Figure 23 shows the JTAG ports on a Samsung i9100, Galaxy S2 PCB and also a JTAG JIG which connects directly to the board without soldering. Once the jig has been connected and the correct wires soldered to it, a JTAG box can be connected. (Reference)

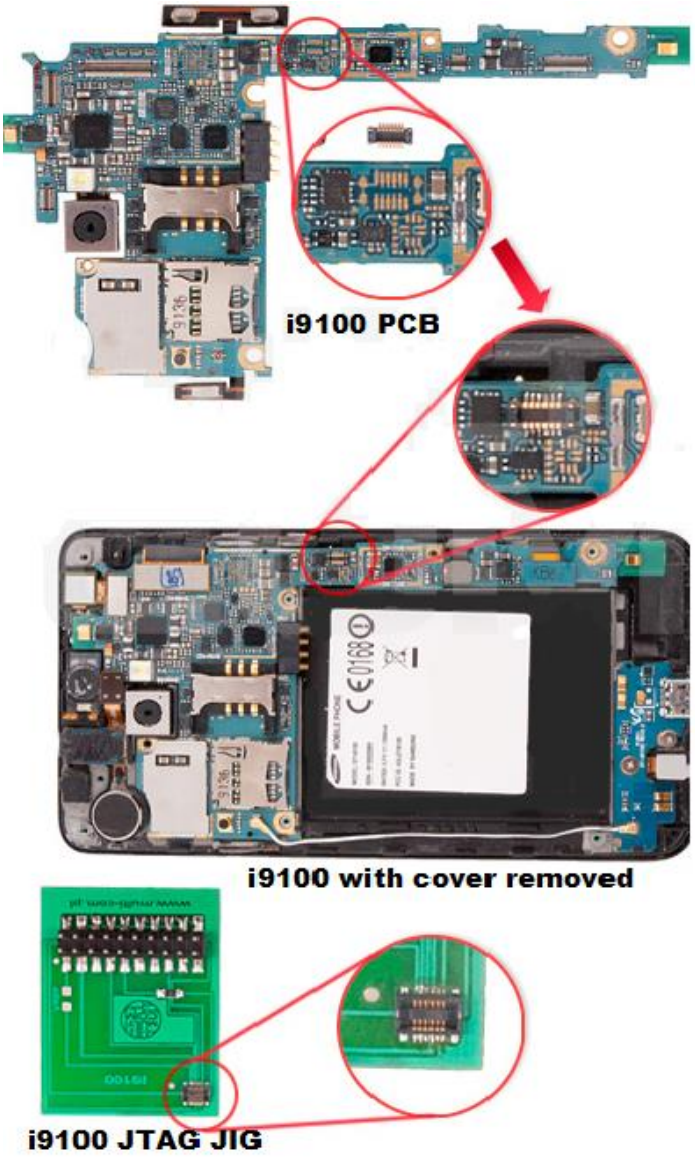


Figure 23: Samsung Galaxy S2 i9100 JTAG Pins

GSM-Technology (n.d.)

A JTAG TAP exposes test signals and most mobile devices include the following:
1. TDI - Test Data In
2. TDO -Test Data Out
3. TCK - Test Clock
4. TMS -Test Mode Select
5. TRST -Test Reset
6. RTCK - Return Test Clock

JTAG acquisition is an area of smartphone forensics, used to physically extract information from smartphones, when software based extractions cannot be used. Using specialized equipment it is possible to recover the full contents of Flash memory from Android devices even when they are security locked or damaged. It is necessary to have detailed information about the specific model of smartphone in order to capture the full contents of Flash. Device manufacturers have JTAG schematics, but they are generally considered company confidential and are not released to the public. A major difficulty when attempting to use JTAG to extract data from a device is identifying the function of each JTAG TAP. This can be done by measuring the voltage at each pad.
There are several challenges that can prevent successful acquisition of Flash memory via the JTAG interface:

- A customized JTAG adapter (JIG) is needed which will vary depending on the device.
- Not all test circuits are universal and may require different parameters to establish communication.
- Some devices prevent access to the JTAG interface for security reasons, making the JTAG TAPS inaccessible
- Commercial forensic tools may not support parsing data structures in Flash memory.

The Riff JTAG Box

RIFF BOX uses standard ARM 20-pin JTAG interface connector:

1	VCC	2	N.C.
3	TRST	4	GND
5	TDI	6	GND
7	TMS	8	GND
9	TCK	10	GND
11	RTCK	12	GND
13	TDO	14	GND
15	NRST	16	GND
17	N.C.	18	GND
19	N.C.	20	GND

Labelled Riff Box JTag Connector

The RIFFBOX RJ-45 Connector:

1	4.2V
2	UART TX
3	UART RX
4	UART TX2
5	MBUS
6	PROBE
7	BSI
8	GND

Figure 24: The Riff JTAG Flasher Box

Some companies make custom connectors which support a specific device and simplify the connection to the pads by placing the PCB between two jig boards with pogo pins. The pogo pins make contact with the JTAG pads on the PCB and can then easily connect to the flasher box. However, experienced engineers may find that soldering the leads directly to the PCB provides a more stable connection.

Once the leads are connected to the appropriate pads, power must be applied to the board to boot the CPU. Each CPU manufacturer publishes the reference voltage for their

hardware and this voltage must not be exceeded. Some flasher boxes provide an option for managing the voltage but in general the power should be managed through an external power supply with a built-in digital volt meter to ensure accuracy. Once the board is powered on, the flasher box software has the ability to perform a full binary memory dump of the NAND flash. However, the connection is serial and takes a considerable amount of time.

Despite all of the complexities, if the JTAG technique is executed properly, the phone can be reassembled and will function normally with no data loss. Figure 11 shows a RIF Box connected to a Samsung Galaxy using a Jig board. The Riff box connects to PC and software selects the phone and action to be carried out. The RIFF Box can also be used to repair corrupted boot-loaders and 'Bricked 'phones also.

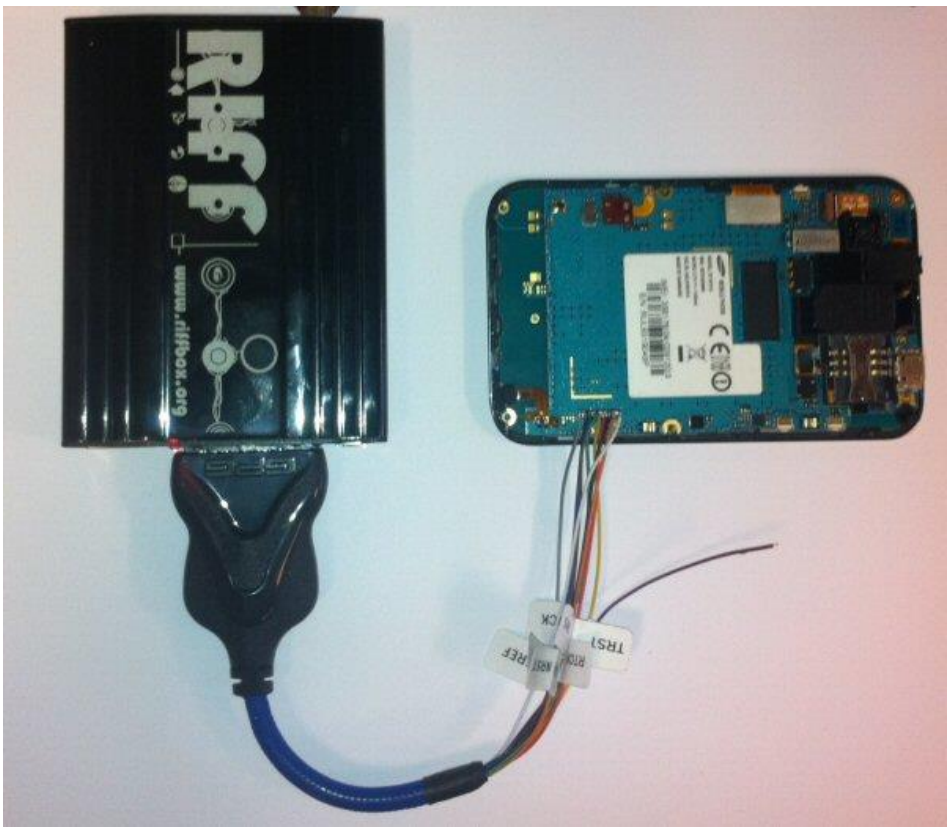


Figure 25: Riff JTAG Box connected to Samsung Galaxy Phone

Though JTAG is an option for extracting data from an Android device's NAND flash, it is very difficult and should only be attempted by qualified personnel with sufficient training and specific experience soldering small PCB connections. Errors in soldering to the JTAG pads or applying the wrong voltage to the board could not only disable JTAG but can also seriously damage the device. For these reasons, JTAG is not typically the first choice for a physical forensic image of an Android device.

If a device cannot be read with JTAG or imaged using software based techniques, then the chip-off method is certainly a valid data extraction method. Chip-off forensics is a data extraction and analysis technique where the NAND flash chips are physically removed from a subject device and examined externally using specialized equipment. Chip-off forensics is a powerful capability for the recovery of damaged devices and also circumvents pass code protected devices. This removal process is generally destructive — it is quite difficult to re-attach the NAND flash to the PCB and have the device operate.
There are three primary steps in the chip-off technique:

1. The NAND flash chip is physically removed from the device by either de-soldering it, or using special equipment which uses a blast of hot air and a vacuum to remove the chip. There are also techniques which heat the chip to a specified temperature. It is quite easy to

damage the NAND flash in this process and specialized hardware, and even controlling software, exists for the extraction.
2. The removal process often damages the connectors on the bottom of the chip, so it must first be cleaned and then repaired. The process of repairing the conductive balls on the bottom of the chip is referred to as reballing.

3. The chip is then inserted in to a specialized hardware device so it can be read. The devices generally must be programmed for a specific NAND flash chip and support a number of the more popular chips already.

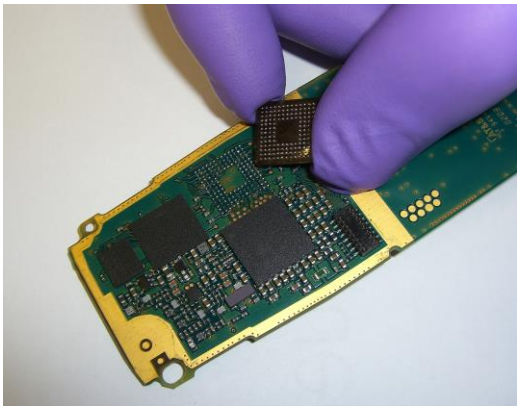


Figure 26: Chip Forensics

http://blog.forensicts.co.uk/?p=336

Now that chip has been read, a physical image of the data stored on the NAND flash chip has been created. Although the chip-off process is quite effective, there are some negative points to consider. The cost of the equipment and tools is expensive and an examiner must again have very specialized training and skills. There is always the risk that the NAND flash chip will be damaged with chip-off, generally in its removal from the PCB. A clean room with protections from static electricity is also desirable.

After a physical dump of the complete memory chip has been obtained, the search for the password lock can begin. To find the hashsums of the passphrase keep the following points in mind:

- The dump of the memory is broken into chunks of 2048 bytes
- The password.key file contains two hashes, together 72 bytes long:
- a SHA-1 hash (20 bytes long)
- a MD5 hash (16 bytes long)
- These hashes only contain the characters 0-9 and A-F
- The following 1960 bytes of the chunk are zeros
- The remaining 16 bytes of the chunk are random

As SQLite stores all data in plain text the lockscreen.password_salt is the first target. A string search in the dump, should recover the lockscreen.password_salt string. Once the string is found, there is a rule-set to identify the salt;

- The byte directly in front has to be between 0x0F and 0x35. This byte (LengthByte) represents the length of our salt.

- In front of this byte, there has to be a byte with 0x3D (indicates a serial type representing a string with a length of 24). This is the length of the string discovered from the initial string search.
- In front of this byte has to be a zero byte

If the rule-set applies, this is the right position in the dump and the salt can now start to extracted. Decoding the LengthByte calculates the length of the salt which has to between 1 and 20 bytes. Next extract this amount of bytes directly after the string "lockscreen.password_salt". These bytes are the salt. (Forensic Blog February 2012)

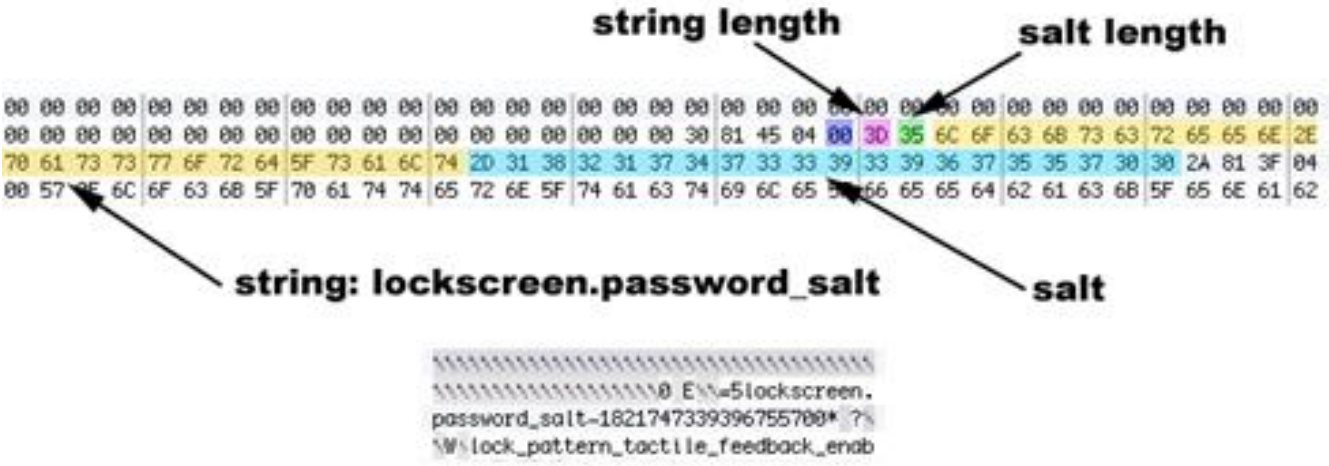


Figure 27: Identifying the lockscreen.password_salt from Raw Hex Dump

Forensic Blog (February 2012)

Once both pieces of information have been acquired (hashes and salt) it is time to start the brute force attack;

```
Santoku# python bruteforce_stdcrypto.py /home/Forensics/ password.key /home/Forensics/ lockscreen.password_salt key
```

3.8 Capturing RAM

Rootfs and tmpfs partitions are mounted in RAM therefore can be captured using Lime (Linux memory extractor). Lime can also capture other useful information, such as browser passwords, images etc. The order of volatility must be adhered to by the examiner when extracting data from smartphones, therefore this capture should be carried out first. If power is lost to the device, important forensic artefacts may be lost. The Lime capture can be filtered to specific applications or processes if the examiner knows what they are looking for in advance. LiME is a Loadable Kernel Module (LKM). LiME ships with a default Makefile that should be suitable for compilation on most modern Linux x86 systems. In order to cross-compile LiME for use on an Android device, it must be cross-compiled using the Buildroot ARM toolchain.

Command to cross-compile

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

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). 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 and not stored locally to device sdcard, which is a forensically sound acquisition method.

For the example below, TCP port 4444 was chosen. First obtain a root shell on the device by using adb and su. To accomplish this run the following commands with the phone plugged into the 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 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, install the kernel module using the insmod command. To instruct the module to dump memory via TCP, set the path parameter to "tcp", followed by a colon and then the port number that adb is forwarding. On the host computer, connect to this port with netcat and redirect output to a file. 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
```

The capture can then be analysed with Volatility on the forensic workstation.

(Lime Forensics March 2013)

Analysing Captured Android Images 4

4.1 Mounting and Examining Ext4 Images

Ext4 images are generally supported in native Linux x86 operating systems. Type the following command to check if EXT4 is supported;

```
Cat /proc/filesystems

Command output here
```

If EXT4 is displayed without nodev in front, it is supported

To mount the user-data EXT4 image type;

```
Mount –t ext4 –o loop, ro, no exec, no load userdata.dd /mnt/ext4
```

Once the partition has been mounted, the investigator can browse through the filesystem. Traditional Linux forensic tools such as Scalpel, Foremost, the Sleuth Kit, Autopsy, the Coroner's Toolkit and PTK Forensics (Commercial GUI for TSK) can be used to recover data. The sleuth kit however needs to be patched to support EXT4. Willi Ballenthin, a consultant at Mandiant, specializing in incident response and computer forensics has developed a set of patches that bring basic support for the Ext4 file system to The Sleuth Kit. The major feature of Ext4 that affects most users is the use of extents that replace indirect blocks. This set of patches supports the new extent structures, and most Ext4 file systems. Until the patches are fully incorporated by Brian Carrier into TSK, they will be developed in parallel and released on http://www.williballenthin.com/ext4/. Support is provided to users and testers of the patches that provide feedback and bug reports. At present the following downloads are available;

- **TSK 3.2.2**
  - r3 (2011-08-21) fix bug affecting older Ext2/3 file systems
  - r2 (2011-07-23) fix sorter tool
- **TSK 3.2.1**
  - r3 (2011-08-21) fix bug affecting older Ext2/3 file systems
  - r2 (2011-07-23) fix sorter tool
  - r1 (2011-06-05) initial patch

To begin setup, build TSK from source to use the Ext4 patches. The patch utility must be installed as well as a compiler toolset such as GCC.

1. **Download TSK source.** Download and extract a version of TSK from http://www.williballenthin.com/ext4/.

2. **Download Ext4 patch.** Download a patch from this website that corresponds to TSK version downloaded.

3. **Apply patch.** From the command line, change directory into TSK source directory. Perform a directory listing, to see the file “Makefile”. Now, run the patch command;

```
root@darragh:/home/sleuthkit-3.2.2#  patch -p1 < TSK-Ext4-r3.patch
patching file tools/sorter/sorter.base
patching file tools/srchtools/sigfind.cpp
patching file tsk3/fs/ext2fs.c
patching file tsk3/fs/fs_file.c
patching file tsk3/fs/fs_inode.c
patching file tsk3/fs/fs_types.c
patching file tsk3/fs/ntfs.c
patching file tsk3/fs/tsk_ext2fs.h
patching file tsk3/fs/tsk_fs.h
```

4. **Build TSK.** Configure and install by running the following commands:

```
dmerrick@ubuntu /sleuthkit-3.2.2# ./configure
dmerrick@ubuntu /sleuthkit-3.2.2# make
dmerrick@ubuntu /sleuthkit-3.2.2# make install
```

Windows tools such as X-ways, EnCase and AcessData’s FTK (Forensic Toolkit) Imager can also be used to extract data. (Willi Ballenthin n.d.)

Scalpel

```
dmerrick@ubuntu# sudo apt-get install scalpel
dmerrick@ubuntu# man scalpel (to learn commands)
```

Before Scalpel can be used it is necessary to define file types that Scalpel should search for in /etc/scalpel/scalpel.conf. By default, all file types are commented out, so uncomment the files that need to be recovered:
dmerrick@ubuntu# cd /etc/scalpel/

```
dmerrick@ubuntu# gedit scalpel.conf
GIF and JPG files (very common)
        gif     y       5000000             \x47\x49\x46\x38\x37\x61  \x00\x3b
        gif     y       5000000             \x47\x49\x46\x38\x39\x61  \x00\x3b
        jpg     y       200000000   \xff\xd8\xff\xe0\x00\x10      \xff\xd9
save
```

Scalpel can be used as follows to try to recover the files:

```
dmerrick@ubuntu# scalpel /media/3965-3439/data.dd -o Scalpel_results
        Scalpel version 1.60
        Written by Golden G. Richard III, based on Foremost 0.69.
        Opening target "/media/3965-3439/data.dd"
```

The -o defines the directory where Scalpel will place the recovered files - in this case the directory is named output and is a subdirectory of the directory where we are running the scalpel command from; the directory must not exist because otherwise scalpel will refuse to start.
After Scalpel has finished, a folder called output will be created in the directory from where Scalpel was executed:

```
dmerrick@ubuntu:~# ls -l output
        total 2
        suspicious_image.jpg
        evildeeds.txt
```

The audit.txt contains a summary of what Scalpel has done:

```
cat output/audit.txt
```

Linux Commands

Using the command, redirect the output to the evidence directory. With that you will have a list of all the files and their owners and permissions on the suspect disk.

```
dmerrick@ubuntu#ls -laiRtu > /home/Evidence/File_List/data
```

Get a list of the files, one per line, using the find command and redirecting the output to another list file:

```
dmerrick@ubuntu# find . -type f > /home/darragh/Evidence/File_List2
```

Now use the grep command on either of the file lists for whatever strings or extensions that that are suspected to be on the device;

```
dmerrick@ubuntu# grep -i jpg /root/evid/file.list.2
        ./csc/common/system/wallpaper/lockscreen_default_wallpaper.jpg
        ./wallpaper/lockscreen_default_wallpaper.jpg
```

This command looks for the pattern “jpg” in the list of files, using the filename extension to alert the examiner to a JPEG file. The -i makes the grep command case insensitive. It is possible that the JPEG’s file name has been changed, or the extension is wrong? run the command ‘file’ on each file and see what it might contain.

```
dmerrick@ubuntu/home/darragh/Evidence# cd /mnt/iso/wallpaper/
dmerrick@ubuntu/mnt/iso/wallpaper# file lockscreen_default_wallpaper.jpg
      lockscreen_default_wallpaper.jpg: JPEG image data, EXIF standard
```

The file command compares each file’s header (the first few bytes of a raw file) with the contents of the “magic” file (usually found in */usr/share/magic,* depending on the distribution). It then outputs a description of the file.If there are a large number of files without extensions, or where the extensions have changed, run the file command on *all* the files on a disk (or in a directory, etc.).

The following command would look for the string “image” using the grep command on the file /root/evid/filetype.list7. Note that using grep to look for the images does NOT catch the Windows Bitmap files. Although these are images, the description does not contain the word "image". Creating keyword lists for graphics can help solve this, although there are other ways.

```
dmerrick@ubuntu# find . -type f -exec file {} \; > /home/darragh/ Evidence /FileType_List
dmerrick@ubuntu:~# cat /root/evid/filetype.list
```

The following command would look for the string “image” using the grep command on the file /root/evid/filetype.list

```
dmerrick@ubuntu/mnt/iso# grep image /home/darragh/Evidence/FileType_List
./lib/libfilterpack_imageproc.so: ELF 32-bit LSB shared object, ARM, version 1 (SYSV), dynamically linked, stripped
./lib/libquramimagecodec.so: ELF 32-bit LSB shared object, ARM, version 1 (SYSV), dynamically linked, stripped
./usr/srec/en-US/metadata: PCX ver. 3.0 image data bounding box [21805, 4691] - [25861, 24430], 110-bit 21333 x 530 dpi,
./csc/common/system/wallpaper/lockscreen_default_wallpaper.jpg: JPEG image data, EXIF standard
./wallpaper/lockscreen_default_wallpaper.jpg: JPEG image data, EXIF standard
```

Search for Android applications

```
dmerrick@ubuntumnt/iso# grep .apk /home/darragh/Evidence/FileType_List
./app/Gmail.apk: Zip archive data, at least v2.0 to extract
./app/GmailSpell.apk: Zip archive data, at least v2.0 to extract
./app/GmsCore.apk: Zip archive data, at least v2.0 to extract
./app/GoogleBackupTransport.apk: Zip archive data, at least v2.0 to extract
./app/GoogleCalendarSyncAdapter.apk: Zip archive data, at least v2.0 to extract
```

Narrow the search to Gmail .apk;

```
dmerrick@ubuntu grep gmail.apk/home/darragh/Evidence
/File_Type.data_partition
dmerrick@ubuntu cd /data/com.google.android.gm/cache
/darraghmerrick@gmail.com
     ls -l
     -rw------- 1 10045 10045   20099 Apr 21 18:45 Invitation Plan for
     Theresa.docx
     -rw------- 1 10045 10045   53189 Apr 29 18:11 Just Weddings Booking
     Form.pdf
     -rw------- 1 10045 10045   64652 Apr 21 12:30 Suspicious image.jpg
     -rw------- 1 10045 10045   16252 Apr 20 13:11 Tattoo-Designs-
     Pictures-13.jpg
     -rw------- 1 10045 10045  176927 Apr 21 18:44 Our Wedding
     Candles.docx
     -rw------- 1 10045 10045   31157 Apr 21 22:07 princess-lotus.jpeg
```

These pictures were all sent or received from this Gmail account

```
dmerrick@ubuntu# gnome-open princess-lotus.jpeg
```

Figure 28: Recovered Image from Gmail account

Use the strings command to view the contents of an executable file, without running it;

```
dmerrick@ubuntu:~# strings suspicious.apk | less
```

To view emails, contacts, etc related to the Gmail account;

```
dmerrick@ubuntu# cd /mnt/ext4/data/com.google.android.gm/databases
dmerrick@ubuntu/mnt/ext4/data/com.google.android.gm/databases# strings contacts2.db |less
dmerrick@ubuntu# sqlitebrowser mailstore.darraghmerrick\@gmail.com.db
```

Figure 29: SQLite Database Browser

Search the image using a search list containing words and phrases that linked to the crime the suspect is linked with. Any hits will written to hits.txt

```
dmerrick@ubuntu# grep -abif searchlist.txt /media/Galaxy_Note/dd > hits.txt
dmerrick@ubuntu:~# cat hits.txt
32456::meeting will take place in Trinity Capital Hotel
54400: we aim to target government organisations
```

Use xxd to display the data found at each byte offset. The xxd command line hex dump tool, useful for examining files. Do this for each offset in the list of hits. This should yield some interesting results by scrolling above and below the offsets.

```
dmerrick@ubuntu:~# xxd -s 32456 galaxynote.dd | less

    00236b1: 796f 7520 616e 6420 796f 7572 2065 6e74 Our group aims
    00236c1: 6972 6520 6275 7369 6e65 7373 2072 616e to target govern
    00236d1: 736f 6d2e 0a0a 5468 6973 2069 7320 6e6f ment…This is no
    00236e1: 7420 6120 6a6f 6b65 2e0a 0a49 2068 6176 t a joke...I hav
    00236f1: 6520 6861 6420 656e 6f75 6768 206f 6620 e had enough of
    0023701: 796f 7572 206d 696e 646c 6573 7320 636f your mindless co
    0023711: 7270 6f72 6174 6520 7069 7261 6379 2061 rporate piracy a
    0023721: 6e64 2077 696c 6c20 6e6f 206c 6f6e 6765 nd will no longe
    0023731: 7220 7374 616e 6420 666f 7220 6974 2e20 r stand for it.
    <continues>
```

(Grundy B. 2004)

Using Sleuthkit on Ext4 Image – Deleted File Identification and Recovery

```
dmerrick@ubuntu# img_stat galaxynote.dd
IMAGE FILE INFORMATION
--------------------------------------------
Image Type: raw
```

```
Size in bytes: 11408506880

METADATA INFORMATION
--------------------------------------------
Inode Range: 1 - 696321
Root Directory: 2
Free Inodes: 679402

CONTENT INFORMATION
--------------------------------------------
Block Range: 0 - 2785275
Block Size: 4096
Free Blocks: 1286459

BLOCK GROUP INFORMATION
--------------------------------------------
Number of Block Groups: 85
Inodes per group: 8192
Blocks per group: 32768
```

The fls command can also be useful for uncovering deleted files. The fls command can be run on directory entries to dig deeper into the file system structure (or use r for a recursive listing). By passing the Meta data entry number of a directory, we can view its contents. By default, fls will show both allocated and unallocated files. This behaviour can be changed by passing other options. If the inode is followed by "*(realloc)"*, it means that the file name listed is marked as unallocated, even though the Meta data entry for that particular inode is marked as allocated. The inode from the deleted file may have been "reallocated" to a new file.

```
dmerrick@ubuntu# fls -Frd galaxynote.dd > /home/darragh/Evidence/recovered
(The frd argument will search directories recursively)
        r/r * 458837:  .data/rte/imagecache/00072297-700.jpg
```

The inode displayed by fls for this file is *458837*.
To find the file names associated with a particular Meta data entry use the ffind command

```
dmerrick@ubuntu # ffind galaxynote.dd 458837
*/media/Android/data/com.bskyb.sportnews/cache/SMALL_2936342.jpg
```

The file is deleted, as noted again by the asterisk. Next use istat. Fsstat took a *file system* as an argument and reported statistics about that file system. Istat does the same thing; only it works on a specified *inode* or Meta data entry. Use istat to gather information about inode 458837:

```
dmerrick@ubuntu# istat dd 458837
inode: 458837
Allocated
Group: 56
Generation Id: 2699274615
uid / gid: 1023 / 1023
mode: rrw-rw-r--
Flags: No A-Time, Extents,
size: 6026
num of links: 1

Inode Times:
Accessed:  Thu May  9 09:07:49 2013
File Modified:  Thu May  9 09:07:49 2013
```

```
Inode Modified:  Thu May  9 09:07:49 2013
File Created:    Thu May  9 09:07:49 2013

Direct Blocks:
1836662 1836663

dmerrick@ubuntu# istat -f list
Supported file system types:
      ntfs (NTFS)
      fat (FAT (Auto Detection))
      ext (ExtX (Auto Detection))
      iso9660 (ISO9660 CD)
      hfs (HFS+)
      ufs (UFS (Auto Detection))
      raw (Raw Data)
      swap (Swap Space)
      fat12 (FAT12)
      fat16 (FAT16)
      fat32 (FAT32)
      ext2 (Ext2)
      ext3 (Ext3)
      ext4 (Ext4) Note ext4 support
      ufs1 (UFS1)
      ufs2 (UFS2)
```

In the next step the contents of the data blocks assigned to inode 458837 will be sent to a file for closer examination.

```
dmerrick@ubuntu# icat -f ext4 -r dd 458837 > /home/darragh/Evidence
/inode458837metadata
dmerrick@ubuntu:/home/darragh/Evidence# file inode458837metadata
      inode458837metadata: JPEG image data, JFIF standard 1.01
dmerrick@ubuntu:/home/darragh/Evidence# xxd inode458837metadata |less
dmerrick@ubuntu:/home/darragh/Evidence# gnome-open inode458837metadata
```

Figure 30: Image recovered by the SleuthKit

Or alternatively

```
dmerrick@ubuntu:/home/darragh/Evidence# icat -f ext4 -r dd 458837 |display
```

In this demonstration a jpeg chosen from the recovered files found by the fls command, showed it originated from the sky sports application and it turned out to be a picture of David Moyes. These methods can be used to recover vital evidence from other locations on an EXT4 Android device. The commands used demonstrated the Sleuthkit basics, but far more complex searches can be used to search slack space, physical string

search and obtain allocation status. Please refer to 'The Law Enforcement and Forensic Examiner's Introduction to Linux', by Barry J. Grundy for more detailed examples. (Grundy B. 2004)

4.2 Mounting and Examining YAFFS2 Images

Ubuntu currently does not support YAFFS2. There is a feature request to package the YAFFS2 kernel module, which would provide mount support for the filesystem, but currently to get YAFFS2 mount support it has to be compiled from source. The YAFFS website has instructions for compiling a Linux kernel with YAFFS support (using Precise 32-bit). Yaffs is the most widely used file system which is specialised for Flash memory. It's used in millions of devices, both under Linux and other operating systems. It's available under the Gnu Public Licence – the GPL – and commercial terms.

Building a linux Kernel with YAFFS2 Mount Support

When building a kernel with YAFFS2, either Intel or ARM architecture can be run in a Virtualbox machine and the resulting system is available as a Vagrant box. There is an Intel x86 image available for download from http://www.yaffs.net, but mounting Android Arm images, was unsuccessful for reasons not known, therefore it seems more logical to build the YAFFS filesystem using an ARM architecture. Vagrant is needed to allow the use of custom kernels within Virtualbox and provides a way to get a virtual system up and running quickly.

The next step is determined by the examiner, there are many choices on how to build YAFFS2 such as;

- Using the image supplied by the YAFFS website to build an x86 kernel. Runs with Vagrant and VirtualBox.
- Go to http://archlinuxarm.org/platforms and download Linux for the ARM kernel build. Also runs with Vagrant and VirtualBox.
- Download an ARM V6 image from http://archlinuxarm.org/platforms/armv6/raspberry-pi and run it from the raspberry Pi.
- Compile YAFFS2 into the physical Ubuntu machine and load the modules whenever they are needed.

During this research all methods were successfully built, but the x86 versions were not able to successfully mount an Android MTD partition in NandSIM. This could also be related to the commands for NandSim being entered incorrectly, or down to a number of other factors needed to set up the YAFFS2 environment correctly. Judging by the forums and questions answered by Charles Manning himself, there are very few YAFFS2 investigators and developers who have successfully mounted an extracted YAFFS2 image into Linux, with Nandsim and Nandwrite. Recall the mtd-utils package;

- Nanddump is used to extract the image
- NandSim is used to simulate a Nand chip in RAM, to facilitate mounting the extracted image
- Nandwrite is used to write the image into NandSim or actual flash devices. (Memory Technology Devices n.d.)

Building YAFFS2 into Arm linux for the Raspberry Pi

To build an Arch Arm linux for the Raspberry Pi for example, download the zip file containing the dd image from one of these links provided on the webpage. Extract the zip file to the local Linux machine. Once extracted it is a dd image named archlinux-hf-2013-06-15.img
Write this image to the target SD card. The SD card will need to be 2GB or larger.
Linux
Replacing sdX with the location of the SD card, run:
dd bs=1M if=/path/to/archlinux-hf-2013-06-15.img of=/dev/sdX
Windows
Download and install Win32DiskImager
Select the archlinux-hf-2013-06-15.img image file, select your SD card drive letter, and click Write. Eject the card from your computer, insert into the Raspberry Pi, and power it on.
The Raspberry Pi's USB ports are limited to 140mA. This limitation has been fixed in newer boards; however, you may still run into power issues, which will be discussed in Chapter 5.
The default username is 'root' with a password 'root'. Once booted up, the command-line of a brand-new installation is visible. To make a kernel with Yaffs kernel-build tools for this distribution, the kernel source code, and the Yaffs source code will be needed. Buildroot could also be used to build a kernel targeted for ARMv6. Again it is down to the investigator/developer which methods to use.

Next download Yaffs from Linux terminal;

```
dmerrick@ubuntu# git clone git://www.aleph1.co.uk/yaffs2
dmerrick@ubuntu# ls
```

Start off in the directory where the kernel and Yaffs source are stored. List the branches;

```
dmerrick@ubuntu# git branch
dmerrick@ubuntu# yaffs-armv6
dmerrick@ubuntu# git checkout master -b yaffs-armv6
```

Now a branch has been created, the star indicates that the user is working in the new branch

```
dmerrick@ubuntu# git branch
dmerrick@ubuntu# * yaffs-armv6
```

Now there is a kernel source, but Yaffs needs to be included. Yaffs comes with a script which automates the source changes necessary to include it.

```
dmerrick@ubuntu# cd ../Yaffs
dmerrick@ubuntu# less readme-linux.txt
```

The script used to patch the kernel is

```
dmerrick@ubuntu#./patch-ker.sh c m ~/kernel/source
```

The c tells the script to copy the changes rather than linking them, and the m indicates that the multi-process option is wanted because more than one process might be using Yaffs at one time. Yaffs has now been patched into the kernel. The next step is to configure the kernel build. To avoid a long compile, remove all of the non-essential drivers, and hardware.

```
dmerrick@ubuntu# make menuconfig
```

This opens up a GUI. Navigate through the menu.To include Yaffs enable two things;

- The memory technology device support (MTD).
- The caching block device access for MTD.

The MTD settings are in the Device drivers section. By default it should be turned on. The caching should be on too. Once those are enabled, Yaffs becomes available as an optional file system. Go into File systems, Miscellaneous and enable Yaffs. Yaffs is setup. Next it is recommended to turn off all the things that are not needed in this kernel build. There are a lot of them! Once the config is suitable, exit and save. To build the kernel, follow these steps;

```
dmerrick@ubuntu# fakeroot make-kpkg clean
dmerrick@ubuntu# fakeroot make-kpkg –initrd –append-to-version=yaffs kernel-image
```

A Debian package has now been created. To install it;

```
dmerrick@ubuntu# sudo dpkg –install
```

To test has YAFFS2 been successfully added to the kernel, boot up the machine. From the command line;

```
dmerrick@ubuntu# cat /proc/filesystems

dmerrick@ubuntu# sudo apt-get mtd-utils
```

YAFFS2 should be listed as a supported filesystem. The mtd-utils contains the Nandwrite, Nanddump and NandSim programs. Next create a Yaffs Nand Simulated drive using the following:

```
dmerrick@ubuntu# sudo modprobe nandsim first_id_byte=0xec second_id_byte=0xd3
third_id_byte=0x51 fourth_id_byte=0x15
```

The following are examples of other NandSim input parameters:

```
modprobe nandsim first_id_byte=0x20 second_id_byte=0x33 - 16MiB, 512 bytes page;

modprobe nandsim first_id_byte=0x20 second_id_byte=0x35 - 32MiB, 512 bytes page;

modprobe nandsim first_id_byte=0x20 second_id_byte=0x36 - 64MiB, 512 bytes page;

modprobe nandsim first_id_byte=0x20 second_id_byte=0x78 - 128MiB, 512 bytes page;

modprobe nandsim first_id_byte=0x20 second_id_byte=0x71 - 256MiB, 512 bytes page;

modprobe nandsim first_id_byte=0x20 second_id_byte=0xa2 third_id_byte=0x00
fourth_id_byte=0x15 - 64MiB, 2048 bytes page;

modprobe nandsim first_id_byte=0xec second_id_byte=0xa1 third_id_byte=0x00
fourth_id_byte=0x15 - 128MiB, 2048 bytes page;

modprobe nandsim first_id_byte=0x20 second_id_byte=0xaa third_id_byte=0x00
fourth_id_byte=0x15 - 256MiB, 2048 bytes page;

modprobe nandsim first_id_byte=0x20 second_id_byte=0xac third_id_byte=0x00
fourth_id_byte=0x15 - 512MiB, 2048 bytes page;

modprobe nandsim first_id_byte=0xec second_id_byte=0xd3 third_id_byte=0x51
fourth_id_byte=0x95 - 1GiB, 2048 bytes page;
```

(Memory Technology Devices n.d.)

In the commands below, there's no need to modprobe mtd or mtdblock because the armv6 image has these compiled into the kernel. Experiment with different Nandwrite arguments to successfully write to /dev/mtd0. It will depend on how the image was captured.

```
dmerrick@ubuntu#   Nandwrite –a –r /dev/mtd0 /media/usb/mtdblock5.nanddump
dmerrick@ubuntu# mount -t yaffs2 /dev/mtdblock0 /media/YAFFS2
```

When using NandSim is essential to;

- Enter the correct parameters to load the right size of the image loaded.
- Write the OOB bytes in the OOB-based image, so that the yaffs2 filesystem is correctly loaded.

Building YAFFS2 into a Vagrant Virtualbox image

After you have Vagrant installed, to test the Yaffs box, open a command line/terminal and type:

```
$ mkdir ~/yaffsprecise
$ cd ~/yaffsprecise
$ vagrant box add yaffsprecise /wherever/you/saved/package.box
$ vagrant init yaffsprecise
$ vagrant up
$ vagrant ssh
```

This will open the command line of the Yaffs box. The kernel and Yaffs can be found in the ~/kernel directory (i.e. /home/vagrant/kernel).

The procedure involves setting up Virtualbox, with Vagrant to allow creating of a kernel from on an image mounted on Virtualbox. Once this is done, YAFFS2 needs to be patched to the kernel. Once the kernel is built and booted up, execute cat /proc/filesystems. YAFFS2 is now supported. Now using kernel modules mtd mtdblock and nandsim the image must be loaded into a virtual Nand chip, simulated by Nandsim. (Charles Manning n.d.)

Now mount the image;

```
$ Mount -t yaffs2 –o ro /dev/mtdblock0 /mnt/forensics
```

The sleuth kit will not recognise the yaffs2 filesystem. YAFFS2 uses deleted and unlinked header chunks to mark an object as deleted. Hence, an object is (at least partially) recoverable from a YAFFS2 NAND until garbage collection deletes the object's entire chunk. Information must be carved from SQLite tables. Some Sleuth Kit commands will help gather filesystem information;

- fsstat -f yaffs2 yaffs2-nexus-one-postdeletion.nanddump
- fls -p -r -f yaffs2 yaffs2-nexus-one-postdeletion.nanddump
- icat -f yaffs2 yaffs2-nexus-one-postdeletion.nanddump 3408468

These Sleuth Kit commands are enough however to recover help recover deleted files. Deep YAFFS2 analysis is beyond the scope of this research, but the basics are covered. Recovery

of /data/data app data as previously described can be carried out as previously described in Ext4. Forensic tools such as scalpel, Foremost and X-Ways could be used against a YAFFS2 image to carve images and files from known file headers

4.3 Mounting FAT images

mount -t vfat -o loop, ro, noexec img.dd /mnt

Examine directories paying attention to Downloads, Photos, and DCIM etc. Attempt to carve using traditional methods. Use X-Ways to examine raw meta-data. The Sleuth Kit and autopsy will also work with FAT images as they are well supported by forensics tools.

4.4 Examining Memory Dumps with Volatility

Volatility can be used to recreate the set of commands that would be run on a Linux system to investigate activity and possible compromise from a live data capture. Volatility can recover a number of useful structures:

–Processes
–Memory Maps
–Networking Information

The Volatility Framework is a completely set of open source tools, written in Python under the GNU General Public License, It is used to extract data from RAM. The RAM samples are mounted as a filesystem, allowing the investigator to examine images of processes and tmpfs mounted filesystems. The latest release Volatility 2.3, in June 2013 supports memory dumps from Arch-ARM-Linux (Raspberry Pi) and Android devices. The new features will include;

- New ARM address space to support memory dumps from Linux and Android devices on ARM
- Added plugins to scan linux process and kernel memory with yara signatures, dump LKMs to disk, and check TTY devices for rootkit hooks
- Added plugins to check the ARM system call and exception vector tables for hooks.

The next step will require a working dwarfdump installation. Linux users should try apt-get install dwarfdump or the libdwarf-tools package. First build libdwarf and then build dwarfdump (no make install for either):

```
dmerrick@ubuntu$ tar -xvzf libdwarf-20130207.tar.gz
dmerrick@ubuntu $ cd dwarf-20130207/libdwarf
dmerrick@ubuntu $ ./configure && make
dmerrick@ubuntu $ cd ./dwarfdump
dmerrick@ubuntu $ ./configure && make
```

Build a Volatility Profile

Get Volatility 2.3 or greater and change into the linux directory:

```
dmerrick@ubuntu $ svn checkout https://volatility.googlecode.com/svn/trunk/ ~/android-volatility
dmerrick@ubuntu $ cd ~android-volatility/tools/linux
```

Edit the Makefile so that;

```
obj-m += module.o
KDIR := ~/android-source
CCPATH := ~/android-ndk/toolchains/arm-linux-androideabi-4.7/prebuilt/darwin-x86/bin
DWARFDUMP := /Users/Michael/Downloads/dwarf-20130207/dwarfdump/dwarfdump
-include version.mk
all: dwarf
dwarf: module.c
 $(MAKE) ARCH=arm CROSS_COMPILE=$(CCPATH)/arm-linux-androideabi- -C $(KDIR)
CONFIG_DEBUG_INFO=y M=$(PWD) modules
 $(DWARFDUMP) -di module.ko > module.dwarf
```

Then make the module.ko driver. The output should be a non-empty module.dwarf file.

```
dmerrick@ubuntu $ make
dmerrick@ubuntu $ head module.dwarf
.debug_info
<0><0x0+0xb><DW_TAG_compile_unit> DW_AT_producer<GNU C 4.7>
DW_AT_language<DW_LANG_C89>
DW_AT_name</Users/Michael/Desktop/volatility_android/tools/linux/module.c>
DW_AT_comp_dir</Users/Michael/android-source> DW_AT_stmt_list<0x00000000>
<1><0x1d><DW_TAG_base_type> DW_AT_byte_size<0x00000004>
DW_AT_encoding<DW_ATE_unsigned> DW_AT_name<long unsigned int>
<1><0x24><DW_TAG_pointer_type> DW_AT_byte_size<0x00000004>
DW_AT_type<<0x0000002a>>
. . . .
```

Now combine module.dwarf and the System.map from your android kernel source code into a zip file. Put it in the volatility/plugins/overlays/linux directory of your Volatility package:

```
dmerrick@ubuntu $ zip ~/android-volatility/volatility/plugins/overlays/linux/Golfish-2.6.29.zip
module.dwarf ~/android-source/System.map
            adding: module.dwarf (deflated 90%)
            adding: Users/Michael/android-source/System.map (deflated 73%)
```

Examine the Memory Dump with Volatility

Android is based on Linux so Linux commands can be used to analyze the memory dump.

```
dmerrick@ubuntu $ cd ~/android-volatility/
dmerrick@ubuntu $ python vol.py --info | grep Linux


                        Volatile Systems Volatility Framework 2.3_alpha
                        LinuxGolfish-2_6_29x86 - A Profile for Linux Golfish-2.6.29 x86
dmerrick@ubuntu $ python vol.py --profile=LinuxGolfish-2_6_29x86 -f ~/lime.dump linux_pslist


                        Volatile Systems Volatility Framework 2.3_alpha
Offset     Name                 Pid             Uid             Gid    DTB        Start Time
---------- -------------------- --------------- --------------- ----
0xf3812c00 init                 1               0               0      0x33b04000 2013-02-25 16:42:16 UTC+0000
0xf3812800 kthreadd             2               0               0      ---------- 2013-02-25 16:42:16 UTC+0000
0xf3812400 ksoftirqd/0          3               0               0      ---------- 2013-02-25 16:42:16 UTC+0000
```

(Hale M. Feb 2013)

4.5 Open Source Toolkits and Android Forensic tools.

Santoku and OSAF (Open Source Android Forensics) are two open source toolkits available. This section lists some of the programs and features offered in each.

Santoku

Santoku includes a number of open source tools dedicated to helping mobile forensics, malware analysis, and security testing needs, including:

Development Tools:

- Android SDK Manager
- AXMLPrinter2
- Fastboot
- Heimdall (src | howto)
- Heimdall (GUI) (src | howto)
- SBF Flash

Penetration Testing:

- Burp Suite
- Ettercap
- Mercury
- Nmap and Zenmap (As Root)
- OWASP ZAP
- SSL Strip
- w3af (Console)
- w3af (GUI)

Wireless Analyzers:

- Chaosreader
- Dnschef
- DSniff
- TCPDUMP
- Wireshark
- Wireshark (As Root)

Device Forensics:

- AFLogical Open Source Edition (src | howto)
- Android Brute Force Encryption (src | howto)
- ExifTool
- iPhone Backup Analyzer (GUI) (src | howto)
- libimobiledevice (src | howto)
- scalpel
- Sleuth Kit

Reverse Engineering:

- Androguard
- Antilvl
- APK Tool
- Baksmali
- Dex2Jar
- Jasmin
- JD-GUI
- Mercury
- Radare2
- Smali

Open Source Android Forensics Toolkit

Command Line Tools;

- aimage
- antiword
- sudo autopsy (starts graphical interface for sleuthkit)
- dc3dd - imaging tool
- dcfldd - imaging tool
- ddrescue - recovers dd image files
- exiftool
- extract
- gzrecover
- magicrescue
- md5deep - hashes files and etc...
- missidentify - finds MS executable files without the executable extension
- outguess - stegonography finder
- pasco - browses IE cache files and parses data
- rar
- reglookup - browses windows registry
- rifiuti2 - windows recycling bin analyzer
- safecopy - data recovery tool
- testdisk
- unrar
- viaForensics-AFlogical
- viaForensics-Parse AF Physical
- viaForensics-Remove OOB
- viaForensics-Scalpel config
- viaForensics-Yaffs2-example
- Vinetto
- VIM

GUI Tools;

- 7zip
- Android SDK
    - includes Android 2.3.3 API
    - Anroid SDK Tools
    - Android SDK Platform-tools
    - Extras package
- APKinspector
- AndroGuard
    - Dependencies
    - >=python 2.6
    - networkx
    - ipython
    - python-ptrace
    - pydot
    - chilkat
    - magic
    - pyfuzzy
    - psyco

- pigments

Libraries needed;

- bzip2
- zlib
- xz
- snappy
- sparsehash
- python-dev

- Bleachbit
- BlessHexEditor
- BitPim
- Chromium Browswer - with APK downloader plugin (homescreen)
- Diffmerge
- Firefox
  - removed Diginotar ssl Certs from
  - /etc/ssl/certs
  - Permanent Incognito
  - Adblock Plus
  - No-Scripts
  - Better Privacy
  - Ghostery
  - Browser Protect
  - WOT - Web Of Trust
  - Bookmark Toolbar - mobile-sandbox.com
- ghex
- Guymager
  - Dependencies
    - bsd-mailx
    - libguytools2
    - libparted0
    - postfix
    - smartmontools
    - mailx
    - mailutils
- Jarcompare
- JDGUI
- mdbtools - libraries to access MSAccess DB's
- Nokia - QT SDK (needs installation for apkinspector)
  - Dependencies
    - qt4-qtmake
- PyQt4
- Pydot
- Regexxer Search Tool
- Sleuthkit
- Foremost
- Frugal
- sip-4.12
- SQLite DB Browser

Android Toolkit Development 5

5.1 Introduction

The focus of this chapter is to develop tools to contribute to the field of Mobile Phone Forensics. The aims of the tools are to automate the analysis and imaging processes, while creating mobile Forensic tools to allow extraction and imaging on the move. After research several concepts and ideas were chosen;

1. Perl script – A Perl script with menu for carrying out different tasks such as;
    - 1. Screen Unlock and Logical Extraction
    - 2. Determining Filesystem and perform Physical Extraction
    - 3. Gathering device information, such as Kernel, event logs, API version, Carrier Information, processes running, memory utilisation etc.
    - 4. Attempt to Root the Device (Method is limited to certain smartphones)
    - 5. Perform Logical Extraction
    - 6. Extract photos
    - 7. Extract Downloads
    - 8. Live data Forensics with Lime

2. Raspberry Pi, mini-computer Mobile Forensics Toolkit. The raspberry Pi uses Arm processor and could come in useful when mounting images and testing tools. Existing Android Forensics tools can be gathered and cross-compiled to be installed on this portable computer. It uses a 10000nAh battery and also comes with power supply. The OS build can be burned to an ISO image and distributed

3. Peer to Peer Forensics, using one Android device to extract information from another. This concept uses a USB OTG cable, so that one device can adb onto another and carry out data extraction, run scripts to carry out physical or logical extraction

4. Android IOIO Board and Android Application controller. The IOIO board is an android development board, which can be controlled by Bluetooth from an control manager. An application could be built, to open a serial connection to the droid and read write commands. LEDS could indicate status.

5.2 Android Forensic Scripting

When carrying out forensic examinations time is a key factor. The investigator needs to extract information as quickly as possible. To achieve partition imaging quickly, scripts can be written to automate commands and processes and also to avoid user error when entering the commands. For each script written an explanation and flow chart is shown. Script will be attached in appendices on a cd. Perl was the language the programs were written in.

Scripts written were;

- Root Checker – Check if phone is rooted
- Android Xtract – This is a partition imaging program which will connect to a device with adb. It will push the Android Xtract Toolkit to the /dev partition. It can determine which file-system the device uses, print mounted partitions and write device information to a report.txt. The investigator is given the option to enter the partition to be imaged. Once imaging is complete, the investigator can repeat process and image another partition or another device.
- Remove Android Xtract – Once imaging has finished this program will remove programs and traces left over from Android Extract.
- Swipe Unlock – This program disables the swipe lock and gives user access to the device.
- AFLogical - This short program pushes and installs AFLogical and disables the screen-lock to allow the investigator to open the application and start the capture.
- Remove AFLogical – Once AFLogical has finished this short script will remove AFLogical and Anti-Guard.
- Android Live data Forensics – this program will install Lime and capture RAM
- Get Photos search device for photos and extract if found
- Get Google data – Extract Browser history from device.
- Com.android.providers Extract - Extract all data from com.android.providers.telephony, contacts, calendar, downloads, media, security, settings, applications and userdictionary

There is one main script with a menu that gives the investigator a central location to run any of these scripts. The investigator will asked to choose a script [1 -10] and the script will automatically launched.

From lessons learned during the Logical and physical extraction processes, it is clear that the procedures require a lot of commands, knowledge and routines to successful image a device. The procedure involves setting up an adb connection to the device. Android debugging must be turned on. On the PC, adb must be installed and a vendor list created and stored in udev rules. Adb should be copied to /usr/bin, so that it can be run globally from within the Linux terminal. The tasks the script needs to automate are;

1. Detect if an adb device is connected and troubleshoot if device does not successfully connect.

2. Once connected, it must determine which filesystem the device uses. If filesystem can not be detected, warn USR and give further choices.

3. If filesystem is yaffs2, go to YAFFS2 imaging procedure. The imaging procedure will;

- Ask user to enter a partition path to be imaged.
- Calculate md5sum of the partition.
- Execute nanddump command and ask user to open a new window to save the netcat file.
- Ask user to calculate the md5sum of image and compare to original. If md5sum is different start imaging again. If the md5sum is the same continue.
- Give user option to image another partition.
- Give user option to image another device
- End program

3. If filesystem is ext4, go to ext4 imaging procedure. The imaging procedure will;

- Ask user to enter a partition path to be imaged.
- Calculate md5sum of the partition.
- Execute dd command and ask user to open a new window to save the netcat file.
- Ask user to calculate the md5sum of image and compare to original. If different reimage, If same continue.
- Give user option to image another partition.
- Give user option to image another device
- End program

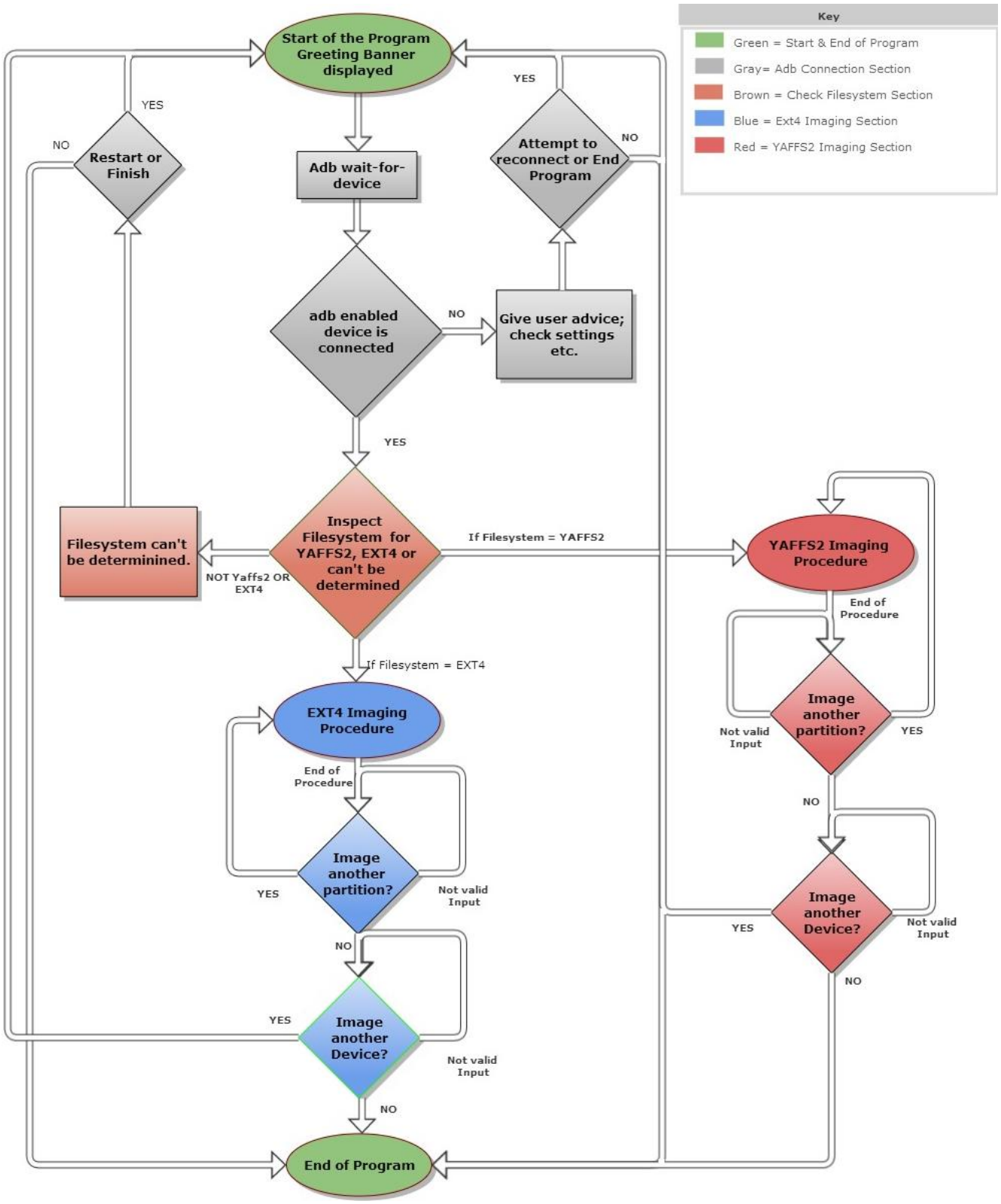


Figure 31(a): Flowchart of Android Extract Perl Program

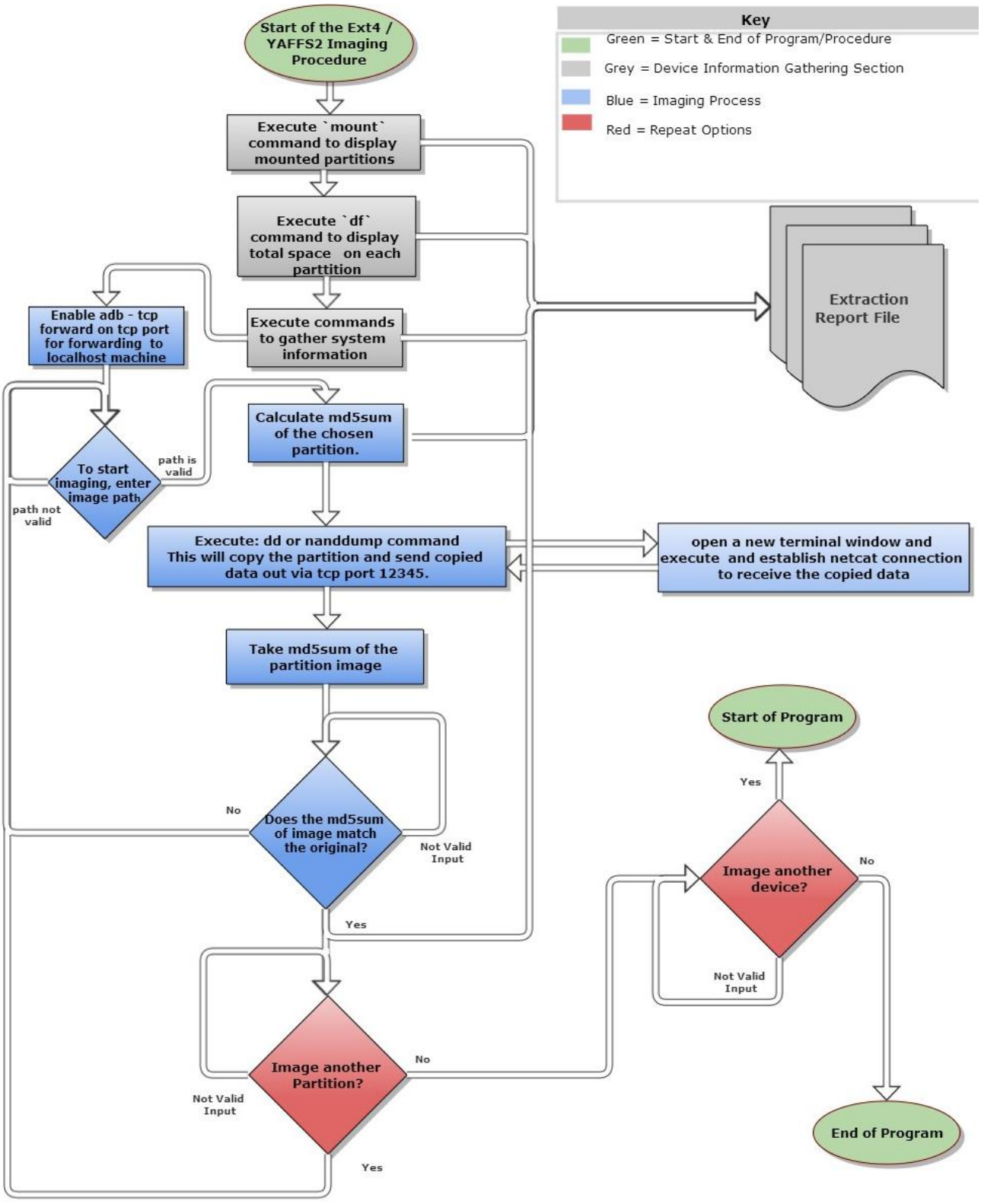


Figure 31(b): Flowchart of Android Extract Perl Program

All of the scripts are listed in the appendices and attached to the printed dissertation on a cd or attached as a directory with the digital copy. Extracts from the finished script can be seen below. It is written in Perl. The comments in green explain the commands. This script achieves all of the actions outlined in the flowchart. It also covers all possibilities such as invalid input, restarting program, ending program, determining filesystem and imaging the partitions.

This section of script establishes an adb connection with the android device;

```
sub connectivity #  This subroutine will start adb and attempt to connect to Android adb enabled device and
                    also determine whether the Device uses an ext4 or yaffs2 filesystem.

   {
            }
   my $wait =`adb wait-for-device`;          # Waits for adb connection.
   my $con1 = `adb devices`;           # Display serial of connected droid.
   my @con2 = split(" ", $con1);
   print " \n @con2 \n";
   if($con2[5] !~ /device/) { # This process troubleshoots no adb connection
                                          (when adb is turned off)!
   print "\n No adb connection! \n Please check that usb is connected to the device and that adb is enabled on
            it.";
   print "\n The local machine has a valid adb vendor list in location /etc/udev/rules.d/51-android.rules\n";
   print "\n Execute 'lsusb' command from Linux terminal to show connected usb devices\n";
   goto error;           }
   if($con2[5] =~ /device/) {
   print "\n adb device successfully connected :-)";
                        }
```

This section determines if the filesystem is YAFFS2 or EXT4;

```
   my $fs = `adb shell 'mount'\n`; #store mount data in an array
   my @fs2 = split(" ", $fs);
   print " Checking filesystem type: \n ";
   foreach (@fs2)
    {
    if ($_ =~ /ext4/) {  # Examines mounted partitions for the presence of ext4 filesystem.
   goto Ext4_Imaging;       }   # Jump to ext4 imaging subroutine.
      elsif ($_=~ /yaffs2/) { # Examines mounted partitions for the presence of yaffs2 filesystem.
   goto yaffs2_Imaging;              }  # Jump to yaffs2 imaging section.

   }
    print " Could not determine filesystem! yaffs2, ext4 and FAT are the only filesystems supported by this
            program\n ";
   goto end;
```

The next section will ask the user to enter the partition they want to image and it will begin imaging the partition with dd or nanddump and pipe the output through netcat to the local machine.

```
   print " \n Which mounted partition is to be imaged? \n Enter full path e.g. /dev/block/mmcblk0p13 \n";
   my $partition = <STDIN>; #Enter partition to be imaged.
   chomp $partition;
   if ($partition !~ /^\/dev\/block\/mmcblk0/) {
   print " You entered $partition \n The file path you have entered is not valid! \n  The file path must contain
          /dev/block/mmcblk0p[n]\n";
   $num = 2; # A delay to give user a chance to read greeting.
   while($num--){
   sleep(3);
            }
   goto Ext4_Imaging;
                        }
   print "Calculating md5sum of $partition will now start\n ";
   my $md5sum = `adb shell su -c "md5sum $partition"`; #md5sum to compare later to image md5sum.
   print " $partition \n $md5sum\n";
   print " \n Open a new terminal window. \n To save the image Enter:\n nc 127.0.0.1 54321|pv >FILENAME.dd
            \n"; # Save the nc data to pc.
   print "\n If netcat won't connect to droid, execute command 'netstat -at' to view connections.\n"; # netstat
troubleshooting message to user.
   my $dd = `adb shell su -c "dd if=$partition bs=4096 | nc -l -p 12345"`;# Execute dd
```

5.3 Raspberry Pi Android Forensics Kit

This project was built successfully. The development started by purchasing a Raspberry Pi, type B Board. Other peripherals were needed such as protective case, 32GB sdcard, HDMI cable, AV cable and USB Power Supply. Raspbian Wheezy was chosen as the OS for the Raspberry PI. Once the OS was configured to build a mobile battery powered the following parts were purchased;

- Raspberry Pi, Model B, Mini Computer
- 7 Inch TFT LCD Car Monitor Screen Display
- Rii 2.4GHz Wireless Keyboard Touchpad
- Anker 12v/5v Battery
- Powergen 5v Battery
- Advent 4 port USB Hub
- Gamestop Sony PSP Carry Case
- USB Cooling Fan (for R Pi)

Initial Setup

The development started by purchasing a Raspberry Pi, type B Board. Other peripherals were needed such as protective case, 32GB sdcard, HDMI cable, AV cable and USB Power Supply. Kali Linux was chosen as the OS for the Raspberry PI, due to the fact that Raspbian Wheezy crashed losing all data several times Kali is a complete re-build of BackTrack Linux, adhering to Debian development standards. All-new infrastructure has been put in place with all tools being reviewed and packaged. Kali supports ARM installations for both ARMEL and ARMHF systems. Kali Linux has ARM repositories integrated with the mainline distribution so tools for ARM will be updated in conjunction with the rest of the distribution. Kali includes more than 300 penetration testing tools; hence it was the best choice for a portable Forensics workstation.

To download and install Kali the following steps were taken;

1. Download Kali Linux from http://www.kali.org/downloads/
2. Extract the image file "distribution-name.img" from the downloaded .zip file.
3. Insert the SD card into the SD card reader and check what drive letter it was assigned.
4. Download the Win32DiskImager utility and extract the executable from the zip file. Run the Win32DiskImager as Administrator.
5. Select the Kali Linux image file and select the drive letter of the SD card in the device box. Click Write and wait for the write to complete.

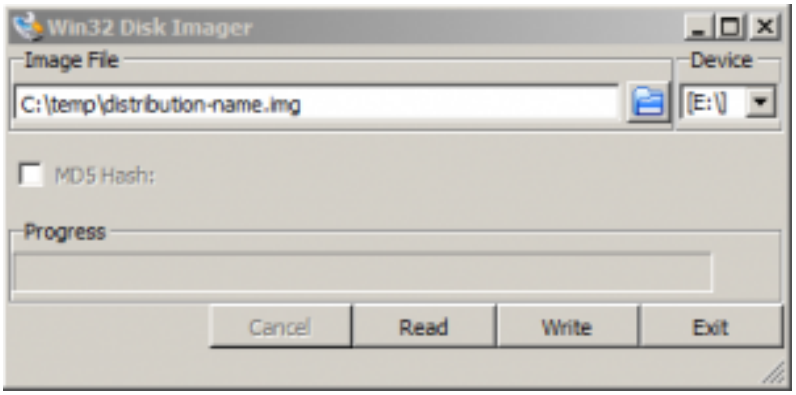


Figure 32: Win32 Disk Imager

6. Exit the imager and eject the SD card. The sdcard is now ready to plug the card into the Raspberry Pi. Turn on the Raspberry Pi. It will boot up. The default password is;

- Username: root
- Password: toor

Once onto terminal prompt enter, to start OS;

```
raspberrypi$ startxfce4
```

Once onto the Desktop, Open File Manager and copy the following files to home directory;

- Android Xtract (Perl Script for imaging)
- Antiguard.apk
- nanddump
- Brute force python script
- AflogicalOSE.apk
- 70-android.rules (Device list)
- Volatility (Is now included in Kali, so not needed)
- Lime.ko

```
dmerrick@ubuntu# cp home/pi/adb /usr/bin (To run adb from anywhere in the Linux terminal)
dmerrick@ubuntu# cp 70-android.rules /etc/udev/rules.d/ (device list needed for adb)
```

Yaffs support is achieved by compiling YAFFS from source and adding it the kernel. This is explained in section ‘4.1 Mounting and Examining YAFFS2 Images’.

There are some power issues with the raspberry Pi. If the correct supply is not used, it will lose power and restart. The diagram in Figure x: shows how to test power. TP1 and TP2 read the voltage. Voltage should not be less than 4.5V. If power drops below 4.2, the Raspberry Pi may shut down. The USB Hub which is needed for connecting all peripherals, needs to be powered externally as this will cause too much of a current draw and cause power failure. Two AC power supplies, two 5v Batteries or a combination of each must be used to power the Raspberry PI. In this case a Sony PSP power supply was altered to a micro-usb interface. It has a 2000mA rating, which is sufficient. When on the move, the Anker is capable of providing steady power. Another issue is keeping the Raspberry Pi cooled. A fan should be used to prevent it from overheating

Figure 33: Raspberry Pi Voltage test Points

The items shown in Figure 34 were used to build the Raspberry Pi portable Forensics kit

Raspberry Pi, Model B, Mini Computer

7 Inch TFT LCD Car Monitor Screen Display

Rii 2.4GHz Wireless Keyboard Touchpad

Anker 12v/5v Battery

Powergen 5v Battery

Advent 4 port USB Hub

Gamestop Sony PSP Carry Case

USB Cooling Fan (for R Pi)

Figure 34: Components of Raspberry Pi Android Forensics Kit

It is important for easy of transport and mobility that all items fit into the case. Figure 35 shows all components, packed into the case for transport. The screen is screwed onto the case, all other parts fit in comfortably. The measurements of the case are 28cm width x 17cm long x 7cm Height.

Figure 35: Raspberry Mobile Workstation packed in case

The photograph in Figure 36 shows the kit in use. The Raspberry Pi is powered by 5v output from the Anker battery pack. The screen is powered by the 12v output from the Anker battery pack. The hub is powered by the Powergen 5v battery pack. Internet is connected by DLink WiFi USB receiver. Keyboard & mouse is connected by USB Bluetooth Dongle. The battery life is approx 1.5hrs. This will be more if the monitor is turned off during imaging process.

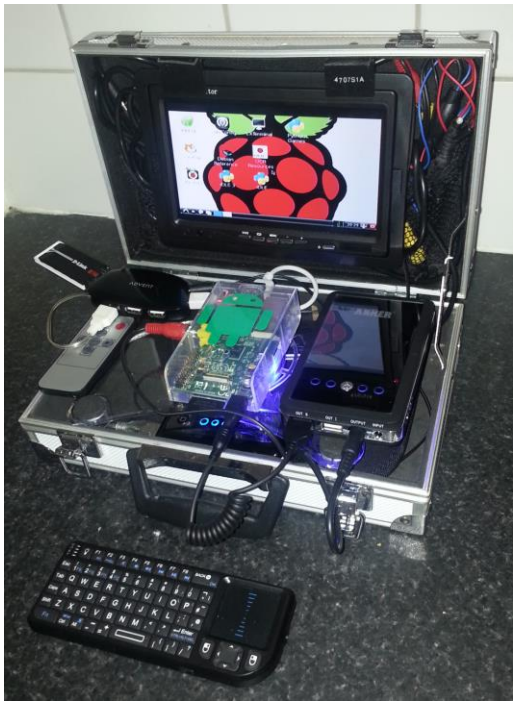


Figure 36: Raspberry Pi Mobile Workstation, ready to use

5.4 P2P Android Forensics

Through the use of the scripts created in section 5.1, installing ADB cross-compiled for the ARM architecture and connecting to the suspect device with a USB OTG cable, it is possible to use an Android smartphone to capture physical and logical images of another Android smartphone. In order to do this a copy of adb built for arm architecture must be pushed to the device and installed in /usr/bin to run from anywhere on the device. If the environment on the investigators Forensics phone is set up correctly and is it connected to the target device with a USB OTG cable, it will have the ability to download files and issue adb and shell commands as if logged on with computer. It took approx 45 seconds to copy over 62MB of com.android.google.* databases and user_prefs files from encrypted HTC Wildfire S to Galaxy Note 2. Also thanks to sl4a (Scripting Layer for Android) the Perl scripts can be executed directly from one Android device to another. A USB OTG (On the Go) cable must be used to allow host connection. It joins pins 4 and 5 of the usb cable to ground.

Making a USB OTG cable

In order to connect 2 Android devices together, a micro USB OTG (On the Go) cable is needed. To make an OTG cable, slice open the micro USB connector end using a Stanley knife. Cut the outer sleeve into two halves and remove it, to reveal the wiring and connector inside. Keep the outer sleeve as it will be glued back together after the work is done.

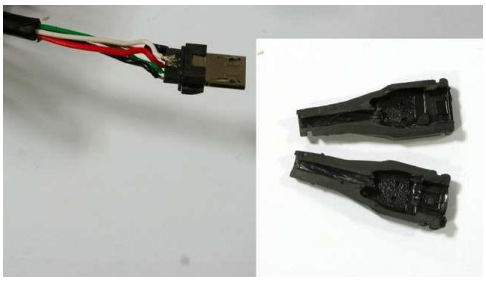


Figure 37: Making a USB OTG Cable

Francis D'sa (June 2012)

After the sleeve is taken apart the connector leads are revealed, five leads and not four. The usual four are power, (+) data, (-) data and ground, while the non-connected lead is sense. This lead needs to be grounded before connecting the cable for the phone to switch to OTG mode and sense a USB device connected to the interface.

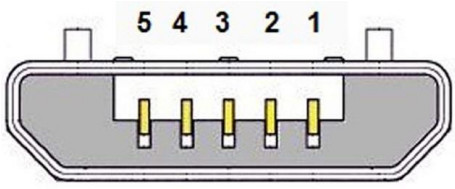


Figure 38: Male Micro USB Connector

Francis D'sa (June 2012)

The pin-out diagram for the micro and mini USB connector is shown below;
Pin 1: VCC
Pin 2: data
Pin 3: data
Pin 4 Not connected / unused
Pin 5: ground

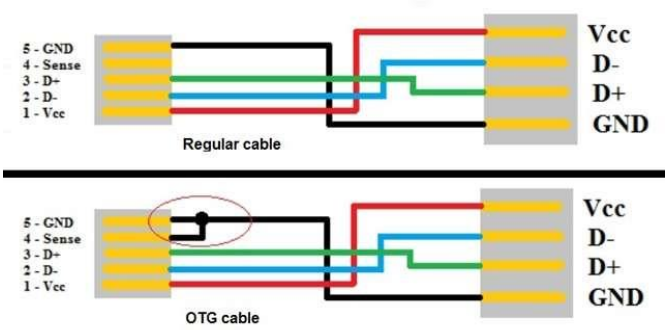


Figure 39: USB OTG Cable Wiring Diagram Francis D'sa (June 2012)

In order to get the phone to go into OTG mode, short Pins 4 and 5 by soldering them together or soldering two wires to each of the pins and leading those outwards from the connector, which can then be soldered to a small switch. Using the switch, it is possible to switch the cable between normal and OTG whenever needed. To short it permanently, cut off the connector at the other end (The Type A Male USB connector) and solder a Type B Female connector to accommodate a USB device. If wanted select a male to female USB convertor at that end.

A small switch can be connected to the leads at Pin 4 and Pin 5 to change between regular and OTG purposes. Next glue the connector sleeves back carefully using hot glue from a glue gun. Now the other end of the cable with a male USB connector needs to be converted into a female. For this use a USB rear panel connector of a desktop PC. Solder the wires of the USB connector to create a USB female-to-female convertor. Once done, an OTG cable is ready for use. (Francis D'sa June 2012)

So that the phone is not damaged, use a multimeter to double check for any unwanted cable shorts which may have occurred during the soldering process. Lastly, connect the OTG cable to the Android phone and test by connecting to another Android phone. If the cable is made correctly, the slave Android will begin to charge from the host device. Execute the 'adb shell' command and it should open a shell terminal on the slave phone. To reduce soldering a Micro USB Host Cable Male to USB Female OTG Adapter can be connected to the micro usb cable with pins 4 and 5 shorted, as seen below. Alternatively if using this cable for actual crime investigation buy a factory made cable from 'The Hakshop' at http://hakshop.myshopify.com/products/micro-to-micro-otg

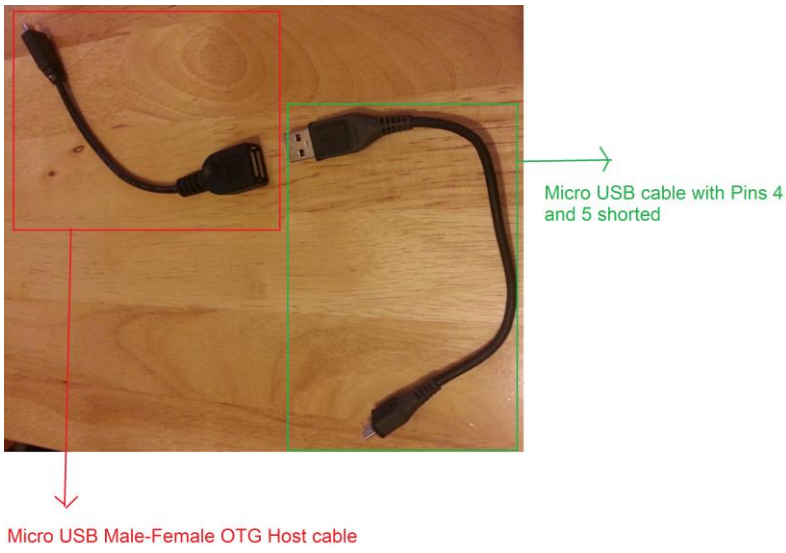


Figure 40: The USB OTG finished product

```
dmerrick@ubuntu# adb install sl4a-r6.apk
```

- Create a directory on the Android device.
- Make a directory on the Android device.
- android# mkdir /sdcard0/sl4a/scripts

```
dmerrick@ubuntu# adb push <script>  /sdcard0/sl4a/scripts
```

- Open the sl4a application. Select menu >view > Interpreters > add > Perl 5.10.1.

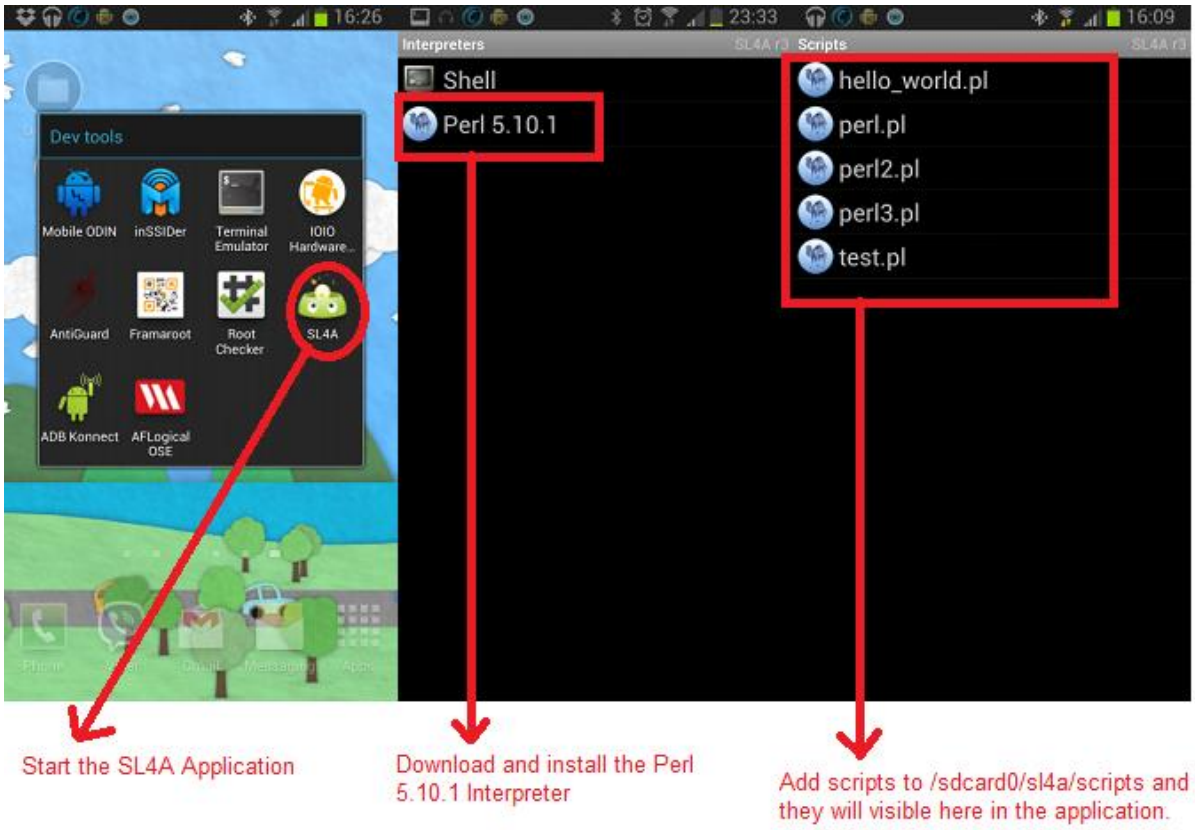


Figure 41: Installing Perl Interpreter on Android

At this stage if the scripts are run, compiler errors will shown. The 'use Android;' statement must be defined at the start of the Android script in order for it to run from an Android device. Also exclamation marks caused errors. In the script the command;
`adb shell su –c "mount"` needed to have exclamation marks removed so that it is written as;
`adb shell su –c mount`

Also the sub-procedures that were originally in the script needed to be removed. The 'use Carp;' and 'use warnings;' statement are also needed in the Perl script, to debug modules and to find script compilation errors. Now an error shows;
"can't execute /bin/sh; no such directory". On Android devices, the shell is located in /system/bin/sh not /bin/sh like it is on most Linux systems. To work around this problem either, use Option 1:

```
android # mount –o remount/ / (This makes / writeable).
android # mkdir /bin
```

Copy bash script from https://sites.google.com/site/ortegaalfredo/android .Click on gdb and gdb-server (6.8) and download android-debug.tbz. Extract the archive, rename bash to sh. and copy the bash file to /bin.

```
android # cp /sdcard0/download/sh /bin
```

Option 2: Create a symbolic link from /system/bin/sh to /bin/sh;

ln -s [TARGET DIRECTORY OR FILE] ./[SHORTCUT]

```
android# ln –s /system/bin/sh ./bin/sh
```

The method chosen is down to the user's preference. Now run the Perl script and it will run.

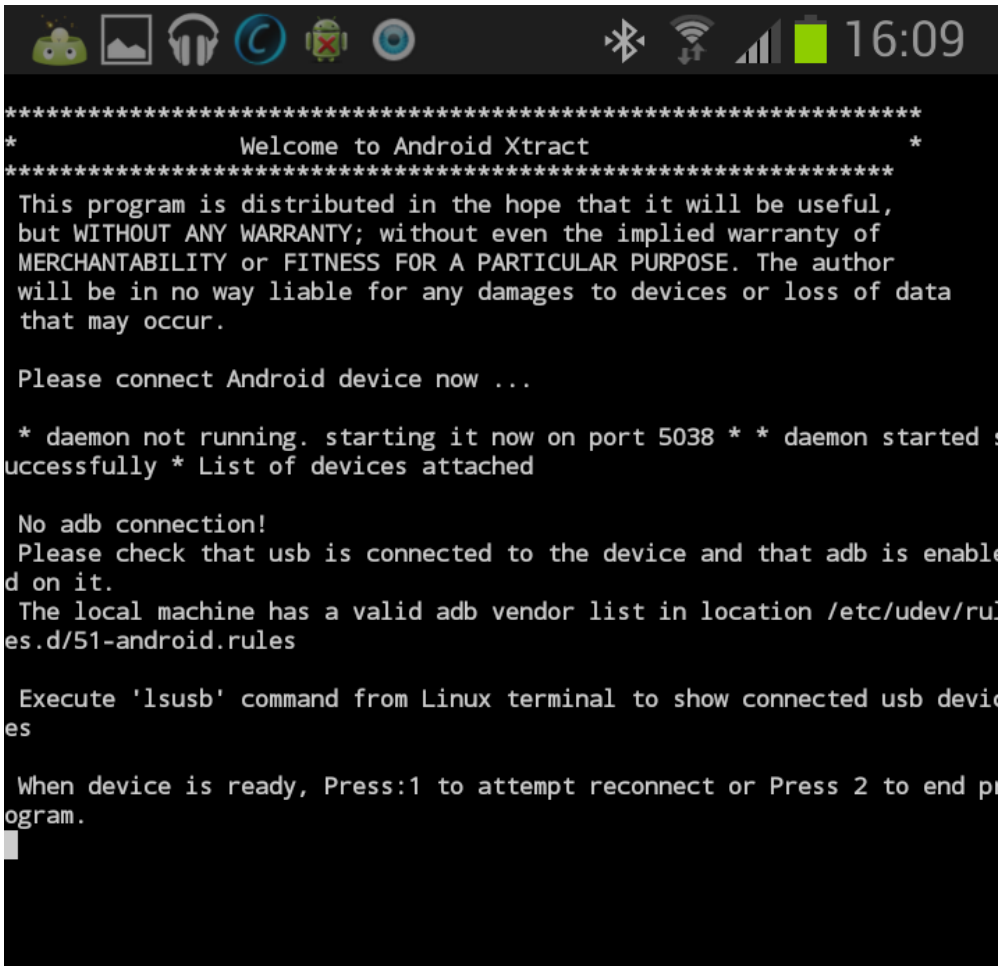


Figure 42: Running Perl Script on Android Device

Logical Extraction using Android P2P

Once the investigators Forensics Android has been configured as described in this section, it is possible to achieve logical extractions very quickly, provided that adb is enabled on the target device.

Step 1: Connect to the target Android device to the Forensic Android device using the micro USB OTG cable.

Step 2: Run the 'Screen Unlock and Logical Extraction script. The script will push Antiguard.apk and AFLogicalOSE.apk. It will then start Antiguard.apk, which will unlock the screen. AFLogicalOSE.apk can now be run, to extract data such as contacts, SMS, MMS and call logs. Once data has been transferred, run the Logical Extract Tool Removal script to uninstall the tools, to remove evidence of intrusion.

Step 2 Alternative: Run other logical extraction scripts, which will gather more evidence such as photos, email history, GPS co-ordinates etc.

Step 3: Disconnect the OTG device and return the device. Obviously there are legal issues with copying data from an individual's phone without permission or warrant, but these steps just highlight that the Android P2P method can operate with speed and stealth.

(Osborn K 2013)

5.5Android IOIO Board

The IOIO is a board specially designed to work with Android 1.5 and later devices. The board provides robust connectivity to an Android device via a USB or Bluetooth connection and is fully controllable from within an Android application using a Java API - no embedded programming or external programmer is needed. The IOIO board contains a single MCU that acts as a USB host and interprets commands from an Android app. In addition, the IOIO can interact with peripheral devices in the same way as most MCUs. Digital Input/output, PWM, Analogue Input, I2C, SPI, and UART control can all be used with the IOIO. Code to control these interfaces is written in the same way as Android app is written with the help of a simple to use app-level library. (IOIO Wiki February, 2013)

The IOIO libraries are loaded into Eclipse. The IOIO board can communicate with Internet/Bluetooth connectivity, touch screen, and a variety of sensors from an Android device, while utilising the Androids powerful processing ability while also allowing connectivity to external Analogue/Digital devices. The true potential of this board has yet to be discovered. Android developers have already built remote control cars, robots, alcohol sensors and motion sensors to name a few projects. Also, using the IOIO does not require any hardware or software modifications to the Android device, thus preserving the warranty as well as making the functionality available to non-hackers. The IOIO acts as a USB host and connects to most Android devices that have USB slave (device) capability. The intended applications of the board are as follows;

1. Act as a JTAG interface and issue JTAG commands from the Android application. Use LEDs to signal connection status etc.
2. Connect to an Android device using serial port (Andropod Interface) and issue imaging commands.
3. Build a rooting device similar to Adam Outler's 'Root any Device' Raspberry Pi. LEDs can signal if rooting is successful.
   *Note these concepts were not completed as they were outside the scope of this project. Building such devices would be a project in itself. Some success was achieved with the IOIO board. The Eclipse environment was set up and some simple apps were developed to read devices and light LEDs etc. The board was mounted on a Galaxy S2 Cradle and wired up to connect using UART with other devices.

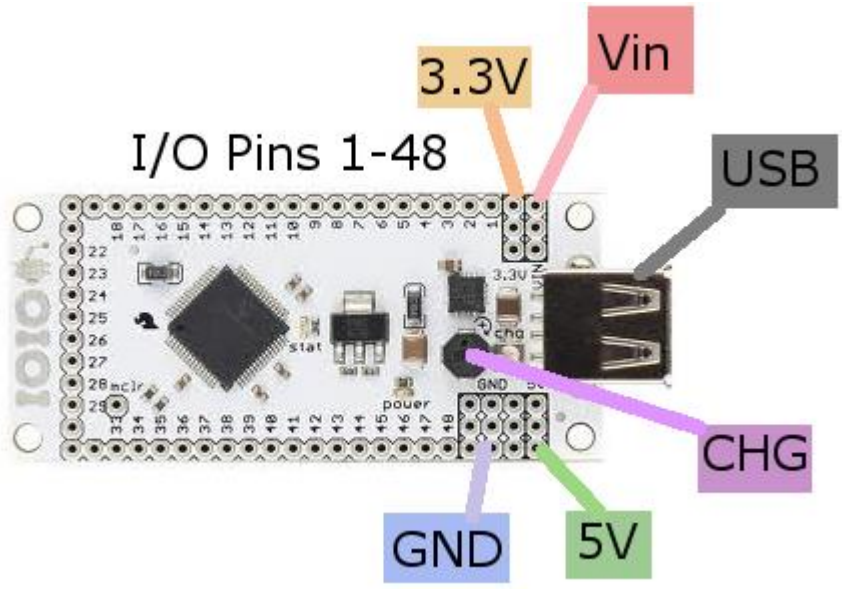


Figure 43: The Android IOIO Board

IOIO Wiki (February, 2013

Figure 44: Using the IOIO Board to extract images from HTC Wildfire S

6 Conclusions

6.1 Nature of main arguments

The initial aim of this research project was;

1. To understand the layers of the Android system, its security measures and how it operates in order to successfully extract data using forensically sound methods and to select the most effective tools to use.

- Solution: Through research, a thorough understanding of the Android stack, boot sequence and filesystem was achieved. In figure 31(a) and 31(b), the flowchart created during this research explains the actions needed to overcome each security feature. The AFPhysical method of extraction is a forensically sound method of physical extraction, which has been written into a Perl script for ease of use and to reduce imaging times. The AFLogical and logical extraction Perl scripts are forensically sound methods of logically extracting data from an Android device.

2. Develop a rooting method for at least one or all Android devices that does not alter the images. There are many rooting methods that will alter the device in different ways. A forensically sound approach is needed that will not alter the phone, allowing the cryptographic hash to remain unchanged before and after the imaging process.

- Solution: A rooting method was not created, as to do so would be a project in itself and outside the scope of this dissertation. Adam Outler's CASUAL cross-platform scripting has achieved universal scripting using a Raspberry Pi, which proves that the concept is possible. However several rooting methods were used. The phones rooted were HTC Wildfire S, Samsung Galaxy S2 and Samsung Galaxy Note 2. ADB Exploits and flashing a recovery partition with root access are the rooting methods most suitable for forensics.

3. In order to contribute to the field of Android Forensics and the open source community this research aims to develop new tools and methods to analyse and automate data extraction from Android devices. Due to the diversity of devices and varied methods for rooting, it can take a lot of time to analyse the devices. Through the use of Perl /Java and shell scripting it is possible to script commands to string search, call Linux/Android commands and write results to SQL tables, so these method could be used to automate some of the forensic processes and determine file system.

- Solution: This was achieved by creating a suite of Android Forensic Perl scripts executable from Windows, Linux or Android. The scripts are explained in Chapter 5 and are attached in Appendices. Also attached is a suite of tools cross-compiled for the ARM architecture. All tools and scripts have been installed on a Raspberry Pi version of Kali Linux. An ISO image of the Kali OS, complete with Android tools and scripts has also been attached in appendices.

4. The script should attempt root access, string search for file system clues, attempt open adb, physical extraction, logical extraction, brute force attacks on swipe codes and passwords. If the device is powered off the challenge will be to obtain the salt stored in the footer and add it to the Encrypted Master Key, in order to break the encryption.

- Solution: This script was not created, but the process of obtaining the salt to perform brute force recovery of the PIN to gain access to an encrypted Android device was successfully carries out.

5. Another project aim was to research Android malware, its growth and how an examiner can detect it. App development and script searching to write the results to SQL tables are methods an examiner could use.

- Solution: Android Malware analysis was researched, but this would be a research project in itself and this section was excluded from this paper.

6. Once methods have been established for a device, they will be documented in an online Android Forensics Wiki, which could be added to by other Forensic Investigators. Access to the site could be restricted by VPN or password authentication.

- Solution: Information sharing is very important among members of the cybercrime investigation community, however if sensitive information were to be posted on public websites, criminals and malware developers may use this information to avoid detection and to create anti-forensics countermeasures to avoid detection. The Android Forensics Wiki has not been created at this time, but will most likely be completed in the near future, if demand.

6.2 Research Structure and Methods

Stage 1: Data Acquisition Stage

The aim of this stage was to gather as much information as possible on the research topic. A platform needed to be established to understand;

- What the challenges are?
- What has already been accomplished?
- What remains a problem for investigators and can be solved by this research in order to contribute to the field of Mobile Phone Forensics.

The research began at the UCD campus library by gathering any books or eJournals related to Mobile Phone forensics. Researching Mobile Forensics online discovered that Andrew Hoog is one of the leaders in the field. His book 'Android Forensics Investigation, Analysis, and Mobile Security for Google Android' was a source of reference throughout the entire research duration. Online research also led to discovering the DFRWS forensics Challenge 2011 which was created in an effort to advance forensic analysis of Android mobile devices. It provided 2 scenarios for examination. The first was carved using the tool 'dd' and the other carved using Nanddump. The winners of the challenge provided a suite of utilities written in Python for extracting information from data acquired from flash memory on Android devices. The tools organised the extracted data to facilitate analysis. The scripts that were written carved data structures from the dd image made of the device. The Nanddump image was mounted into Linux and information from files and databases was obtained.

An interesting discovery from this project was that using 'dd' to acquire data in Scenario 1, did not copy the important information from the OOB (Out Of Bounds) areas of the YAFFS2 filesystem. In order to mount a YAFFS2 image and successfully extract data it, it essential to understand the architecture and filesystem layout. Charles Manning, the Chief Yaffs Architect, provides detailed guides and filesystem information on http://www.yaffs.net/documents. Turning Android Inside Out was a conference and accompanying PowerPoint slideshow released by Ivo Pooters released after the DFRWS forensics Challenge 2011, which highlighted the challenges of forensically analysing a YAFFS2 Android device and how to overcome them. In his analysis he talks about;

- The effects of a rooted phone
- Interesting partitions in the Android file structure.
- Forensic tools that can be used on YAFFS2.
- How to read YAFFS2.
- Enabling YAFFS2 in Linux
- Live Analysis.
- Static Analysis.
- SQLite Databases.
- Recovering SQLite Leaf Pages.
- Data that should have been recovered from the phone, i.e. incriminating documents and malware.

Thomas Cannon an Android Security Researcher, Mobile Forensic and Reverse Engineering expert with ViaForensics has also contributed a lot to the field of Mobile phone forensics. His conference and accompanying PowerPoint slideshow "Into the Droid – Gaining Access to Android User Data" delves into;

- Android Forensic Challenges
- Bootloader Essentials and defeating the bootloader.
- Forensic Boot Images
- Flash and RAM loading
- JTAG Primers
- Serial Debug cables
- Cracking the PIN or Password
- HID Brute Forces attacks.
- Android Encryption and how to crack it.
- Reverse Shells
- Pattern Locks and how they are stored
- File-system Research;
    - Address the lack of support for YAFFS2. YAFFS2 may become obsolete as high end devices now have dual/quad core processors. YAFFS2 only supports single core devices.
- The foundations of Android Security are;
    - Application isolation and permission control – can we control what apps are able to do.
    - Application provenance - can we trust the author.
    - Data encryption – Is the data on the device safe, if it is hacked.
    - Device Access Control – Protect device against unauthorised access control.
- The Android Stack
    - This is what the Android operating system looks like
    - Linux kernel and other layers for applications services etc.
- Android Application isolation
    - By default each app runs in a separate user/group id (fixed for the lifetime of the app).
    - Dalvik VM is the sandbox for apps.
- Default Android permission policy,
    - App can't read/write files outside their own directory
    - Install/uninstall/modify other apps.
    - Access internet
    - Keep device awake (as a Botnet)
- Address Space layout Randomisation
    - Shared libraries on Android are pre-linked. Addresses are fixed for performance reasons.
    - ASLR makes it hard for malicious apps to take advantage of the shared library's fixed addressing.
- Security Concerns
    - Push-based install from Android market
    - Social Engineering

- Firewall
- Encryption of communication
- Compromised platform keys
- App obfuscation
- Protecting bootloader / recovery
- OEM / Carrier OS upgrade cycles

Other Android research topics included;

- Research of Android OS versions
- Research of Open Source V Closed source Android Forensic tools
- How the popularity of Android makes it a target for malware developers and Botnets.
- The diversity in file-systems YAFFS, YAFFS2, EXT 3, EXT 4,FAT 16/32 for sdcard, smbfs (Samba Mountable File-system), rootfs & ramfs (RAM-based file-systems) and the sleuth kit/Volatility support for each.
- Built in permissions
- Escaping the sandbox
- Logical permission Enforcement
- Known Android Malware
- Safeguarding Apps Data
- Data encryption
- Whole disk encryption
- Digital rights management
- Physical access control

` Once a large Android Forensics literature library had been acquired made up of books, e-books, published dissertations, eJournals and WebPages it was then time read all documents and inherit a deep understanding of how Android works and how to forensically analysis it.

Stage 2: Practical Testing

Now armed with the theory from stage 1, it was time to test out the procedures on actual Android devices. A Samsung Galaxy S2 was used as the test case for the EXT4 filesystem. The HTC Wildfire S was used as test case for the YAFFS2 filesystem. The first challenge was rooting the devices. This was risky, because if this is not done correctly, the phone can be 'Bricked", i.e. rendered useless. After researching methods on the XDA Developers website, it became clear that many methods are not geared towards forensics and will erase user data.

The best methods to use for forensics are;

1. Adb Exploits (Which do not alter the device at all).
2. Filesystem Exploits (Which, make minor changes to the filesystem to force root access).
3. Recovery Mode (Flashes a recovery image to the recovery partition and gain root access to the device).

Why is it necessary to root?

Answer: To get access to the system files to create physical image, to access custom ROMs, remove offending apps, get at rootkits.

- Rooting comes at a price. It all falls apart when we allow untrusted code to run as root.
- App isolation, system/app permissions, data safe-guards & encryption, device administration are all compromised

Due to the fact that adb exploits only work on early versions of Android (API 2.1 and below), a custom image was flashed onto the Galaxy S2 using Odin. This erased the user data, so would not be usable for forensics. It did however grant root access for testing. Choosing a rooting method for a device is a difficult task, as there are so many available, depending on the device manufacturer and the Android API

installed. Later when a Samsung Galaxy Note 2 was used for analysis, a Filesystem Exploit was used by rooting the device with the one-click application, Framaroot.

Evaluating ViaExtract

- Licensing only allows for demo version, restricting to 10 entries per section.
- It was downloaded image from viaForensics and installed it on VirtualBox.
- It allowed logical extraction only on free version, but the full version identifies the phone, file-system, logical and physical data extraction.
- All information is viewable in a GUI.

Other tools in Via Extract include Gesture key decode, image storage device, unlock screen, Sleuthkit timeline, Encryption brute force, but none are useable in demo mode.

Santoku is a platform for mobile forensics, mobile malware analysis and mobile application security assessment.

Tools to acquire and analyse data;

- Firmware flashing tools for multiple manufacturers
- Imaging tools for NAND, media cards, and RAM
- Free versions of some commercial forensics tools
- Useful scripts and utilities specifically designed for mobile forensic
- De-compilation and disassembly tools
- Scripts to detect common issues in mobile applications
- Scripts to automate decrypting binaries, deploying apps, enumerating app details, and more

The raspberry pi will not support Santuko because it is an ARM platform. Santuko is only available on x86 platform so this will be installed on virtual box vm on laptop. Other tools tested included;

- Busy-box tools
- Android SDK
- Memory Dump in Android. Ddms, the Dalvik Debugging Monitor server can be used to dump the contents of memory
- LiME Forensics - Linux Memory Extractor
- Cellebrite Mobile data recovery UFED physical (30 day evaluation license)
- The Sleuth kit

Once these tools had been installed and tested it was then time to development new tools and automate verbose commands into easy to run scripts.

Stage 3: Development Stage

During this stage firstly the physical imaging of Android devices was addressed. The most reliable imaging methods were scripted using Perl, so that the examiner can run the script and choose partitions without typing the commands. The filesystem is determined by the script and the correct tools will be used accordingly. The md5sum of the images are taken to ensure the images integrity and filesystem information is copied and all results are compiled in an Examiner Report text document. Other tasks that were scripted included;

- Root Checker – Check if phone is rooted
- Remove Android Xtract – Once imaging has finished this program will remove programs and traces left over from Android Extract.
- Swipe Unlock – This program disables the swipe lock and gives user access to the device.
- AFLogical - This short program pushes and installs AFLogical and disables the screen-lock to allow the investigator to open the application and start the capture.

    - Remove AFLogical – Once AFLogical has finished this short script will remove AFLogical and Anti-Guard.
    - Android Live data Forensics – this program will install Lime and capture RAM
    - Get Photos search device for photos and extract if found
    - Get Google data – Extract Browser history from device.
- Com.android.providers Extract - Extract all data from com.android.providers.telephony, contacts, calendar, downloads, media, security, settings, applications and userdictionary

Mounting the Nanddump images of the Nanddump images is a difficult process, therefore the environment can be set up and loading the modules scripted, so that the investigator doesn't have to waste time researching and understanding mounting YAFFS2 in NandSim.

The portable Raspberry Pi Forensic Workstation was designed in order to have a portable forensic lab, which boots quickly and which also has the benefit of using the ARM processor. This has much compatibility with Android. The Forensics scripts written can also be run on the Raspberry Pi. The Raspberry Pi Kernel, with added cross-compiled Android forensic tools will be copied to an image file, so that an investigator need not waste time building and cross-compiling tools. The OS can be installed directly onto the raspberry and the Android Forensics workstation is ready to go. The image is built around Kali Linux, which contains many pre-installed forensics tools.

The Android IOIO Board has many projects online, but very few related to data extraction. The purpose of this development was to write an Android application on Eclipse to control the board, as an imaging device, write blocker or JTAG device. The capabilities of the IOIO board have not yet shown full potential, with bigger and better projects always being released. The IOIO board can interact with electronic device or other Android devices and commands are issued by an android application from another phone using Bluetooth or USB connection. This idea ended up not being completed, as the java application development was very difficult and the board commands were difficult to learn, but with more time, this idea could be very successful. This IOIO board research could be a dissertation in itself and due to the amount of time it was taking up, it had to be abandoned.

The initial idea for using the IOIO board was to image a device using it, issuing the commands from an attached Android device. The p2p-adb framework, more or less made that idea obsolete. This allowed a target Android device to be connected to an investigator's Android device with a USB OTG cable and by using sl4a Perl script emulator; the scripts could be run directly from the phone, provided it had large enough storage to save the images locally (32 GB Sdcard should suffice). So basically all of the scripts and commands that had been setup on the Linux Desktop and Raspberry Pi are now executable from an Android device.

The cross-compiling of tools proved to be difficult during this project. Buildroot was useful as it automated the building of a toolchain, through the use of a GUI to set up kernel environment. Initially to avoid cross-compiling, it is possible to find pre-cross-compiled tools online, but they are difficult to find and not always available, so building a toolchain is necessary. Buildroot will build a kernel and download all the tools specified for the build in the set up. If forensic tools are specified, they will be downloaded, cross-compiled and made part of the Kernels toolset.

6.3 Research Findings and Results

The research paper addresses the need for more Android and Smartphone forensic tools and support. Computer forensics has evolved from NTFS /FAT/ EXT3 filesystem analysis, which was usually built for an Intel or AMD 32/64 bit desktop or laptop. Methods where an IDE or Sata hard drive could be removed from a suspect device and imaged using a write blocker are no longer feasible for smartphones

and tablets. New methods, tools and forensic standards must be developed in order to assist data extraction from these new devices. The added features of telephone, GPS, VOIP, video recorder, camera etc, to these devices means that possible artefacts and evidence which may be retrievable will differ greatly from a data retrievable from a PC or laptop.

Another factor is how files are deleted on smartphones. On an Android with a YAFFS2 filesystem Garbage collection erases certain blocks in NAND flash to increase the overall amount of free blocks. Valid data that exists in blocks selected for garbage collection will be copied to another block first and thus not be erased. This means that large amounts of undeleted data may exist on the device, which is good from a forensics point of view. The lack of support for YAFFS2 is a problem for forensic examiners. In order to mount a filesystem image in linux, YAFFS2 must be compiled from source and loaded into the kernel as a module. The Nand chip must then be simulated in RAM in order to provide a location to mount the YAFFS2 image. This process is extremely difficult and not well documented. The only information that can be found online explaining how to do this is from threads and forums or the YAFFS website.

Although initially this research set out to develop tools to supportYAFFS2 and methods to mount, recover and analyse YAFFS2, this task proved extremely difficult and time consuming. A failure in the YAFFS 2 filesystem is that it is single threaded and would be a bottle neck on dual /quad core CPU systems. Due to this restriction most modern devices have adopted the EXT4 filesystem to facilitate multiple CPUs. This may lead to decline and eventual extinction of YAFFS2 on smartphones, therefore it may not be necessary to waste too much time on developing tools for YAFFS2. The focus should be directed on the EXT4 filesystem, which looks to be the new Android filesystem, here to stay.

The ACPO Guidelines for Forensic Computing state, that in order for a forensic investigator to examine a smartphone, the device must be handled in accordance with ACPO Guidelines for Forensic Computing**.** The following guidelines from the Association of Chief Police Officers are the principles used during a forensic analysis of computer equipment or mobile phones

- Principle 1: No action taken by law enforcement agencies or their agents should change data held on a computer or storage media which may subsequently be relied upon in court.
- Principle 2: In exceptional 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.
- Principle 3: An audit trail or other record of all processes applied to computer based electronic evidence should be created and preserved. An independent third party should be able to examine those processes and achieve the same result.

If the device needs to be rooted or if tools needed to pushed onto the device via ADB, the investigator must adhere to Principle 2, stating valid reasons for doing so and prove that the methods used were forensically sound and carried out competently. To ensure that devices are handled correctly and that the order of volatility is followed, the flow chart in figure x was created and should be adhered to. Through the use of scripts which create report files and automate commands, the physical and logical imaging can be carried out. It must be noted that the md5sum will not always match, due to the nature of Nand and due to activity on the /data partition. The investigator must make a decision to;

1. Unmount /data and image it. In this case the md5sum of the extracted image will match, but the unmounting of the /data partition itself will trigger changes that will in evidently make changes to the device.
2. Image /data without unmounting it. The md5sum will not match, but the investigator could note the md5sum of image and from this point onwards use it as a source of data integrity.

The state, in which the device is found in, will have a huge impact on how the investigation will be carried out. The investigator should always take the minimum action required. For example if the suspect is suspected of making abusive phone calls and sending threatening text messages, a logical examination should suffice, therefore rooting and physical extraction is not called for. However if the messages and call logs have been erased, a physical examination would be needed to recover the deleted data. Due to the risk involved with altering a live device with no write blocker, all changes and procedures carried out should be well documented in accordance with Principle 3.

Once the EXT4 images have been extracted precede using linux commands, the sleuth kit and forensic tools such as scalpel, x-ways and volatility etc. A SQLite browser can be used to view data stored in databases such as contacts, SMS, MMS and email. The results should consist of;

- A directory of recovered images created by scalpel
- A number of scripts created by the scripts during the imaging process, containing device information, memory dumps, md5sums of partitions and mounted partitions
- Files manually recovered by the Sleuthkit
- SQLite databases extracted from /data partition.

The Android investigation can be carried out using;

- Windows laptop/pc with the SDK and Perl installed.
- Ubuntu laptop/pc with the SDK and Perl installed.
- Raspberry Pi with ADB and Perl installed
- Android with ADB and Perl installed

6.4 Analysis

The main goal of this paper was to simplify and automate Android forensic examinations. This was certainly achieved by creating;

- Flow Charts
- By explaining rooting and extraction tools and methods used to extract data
- Cross-compiling x86 tools to run on Android
- Scripts
- Raspberry Pi Mobile Forensics Workstation
- Samsung Galaxy Note 2, Forensic Workstation.

A problem for investigators is standardisation. Android devices are released by a wide number of vendors and not all of the Vendors are members of the OHA. Android is open source and the source code has been released by Google under the Apache License, allowing the software to be freely modified and distributed by device manufacturers. As a result there are many Chinese Android smartphones /tablets on the market which may not use the same standards, micro USB, and filesystems as European Android market devices. Google does not offer support to these devices and as result some features will not work.

This diversity of handsets, Operating system releases, connectors and hardware makes it extremely difficult for the examiner to reliably extract information. For example the rooting methods outlined in this paper, will only work on a limited number of devices and for certain API releases. The scripts attached in the appendices are functional and useful now, but over the next few years there will be new API releases and there could be changes to

partitions and filesystems, that would require the scripts to be rewritten and updated. All of the findings, procedures and programs from this dissertation will be published as open source, so that other developers, investigators can add to or alter them for their own needs or improve them and give them back to the open source community.

There are wide range commercial tools available for smartphone forensics such as Cellebrite UFED, Logicube CellXtract, Micro Systemation XRY, MOBILedit! Forensic, Oxygen Forensic Suite, Paraben Device Seizure, Radio Tactics and ViaExtract from ViaForensics. Some of these tools have additionally been developed to address the increasing criminal usage of phones manufactured with Chinese chipsets. Cellebrite's CHINEX and eDEC's Tarantula, while other vendors have added some Chinese phone support to their software.

Most open source mobile forensics tools are platform-specific and geared toward smartphone analysis. Examples include the Android SDK platform tools, Katana Forensics' Lantern Lite imager, the Mobile Internal Acquisition Tool, TULP2G, Android Open Source Forensics (OSAF) and Santoku-Linux which includes AFLogicalOSE the Open Source Android Forensics application, by ViaForensics.

In comparison, the commercial tools are far superior to those released as open source. Many of the Open source tools released are light versions which come with limited functionality, such as AFLogicalOSE or Katana Forensics' Lantern Lite imager. Many of the commercial tools use a GUI, which the forensic examiner will push a button and an automated extraction will take place. Although the program will find and display results in an easy to read format, the forensic examiner does not know how this results were achieved. Does the program use open source tools behind the scenes? Or did the program search in slack space for deleted files? These are things the forensic examiner needs to know, in order to carry out a detailed examination of the device. If the sleuth kit was to support YAFFS and officially support EXT4, then perhaps Forensic examinations would use these tools for examination and rely less on the commercial tools.

The smartphone /tablet market is fast changing, therefore forensic tool developers struggle to support these devices. A universal set of forensics tools are needed, that will work on all or most of the devices. Standardisation of filesystems, hardware and Android releases would help forensic examiners. When compared to the Apple iPhone, which used standardised hardware and software and all devices.

Another issue that was not dealt with in this research paper is that many applications can encrypt or hide data in secure vaults. Forensic methods must be produced to decrypt and extract data from such as application. There are many applications such as Hide it Pro, File Hide Expert or Vault-Hide SMS, Pics & videos that will hide pictures, videos, applications, messages, call logs for the phone and messages sent/received by the phone. It should be assumed that any criminal using the phone for criminal activity will use a free application to hide this activity.

6.5 Further Development

- A good knowledge of Java scripting is essential in order to write or analyse Java applications. Creating Java applications to automate forensic tasks or to control an IOIO board, can be done in Eclipse with the ADT (Android Development Tools) add on.
- A JTAG box and collection of Jigs for different devices would be preferable for testing and developing extraction methods.
- Android Malware would be a huge research topic that needs to be investigated. Investigators need to learn how to identify and whitebox, blackbox malware. Perhaps malware could be examined on a virtual Android.
- Development of Open source forensics tools for iPhone, Motorola, Blackberry and other non-Android smartphones/tablets.
- Examination and tool development for Android smartphones using Chinese chipsets.

Android was chosen as the basis for this research topic, due to its growing popularity and market share. Taking into consideration the time restraints, budget for buying equipment and initial Android knowledge, it is fair to say that this research paper achieved reasonable goals. The goal was to become familiar with Android forensic tools and extraction methods and then contribute new tools and methods to the field of mobile phone forensics. With funding, a research team of Digital Forensic graduates and industry professionals, a lot could be achieved in research and development of Open source tools for Smartphone /tablet forensics.

Mobile phone Forensics examiners should know the limitations of software packages and forensic tools, which tools work best on certain devices, be aware of anti-forensics techniques and know where to go for research on various phone types and their potential forensic yield. With the huge increase in smartphone and tablet popularity, the growth of Malware, internet fraud, hacking and criminal activity has followed. Law enforcement, academia and incident response teams must work together and invest time and money into discovering new forensics techniques and tools in order to keep control over these illegal activities. One thing that is for certain, this fast changing communications technology will evolve and change over the next few years and the field of Digital forensics must adapt and evolve with it.

Appendices

Due to the number of scripts and cross-compiled programs, they can be found on an attached CD with the printed dissertation or an archive with the digital copy. The contents of the appendices are as follows;

Scripts

1. Readme.txt - A document outlining the tools and environment needed to run the scripts.

2. Root Checker – Checks if the Android device is rooted.

3. Android Xtract – This is a partition imaging program which will connect to a device with adb. It can push the Android Xtract Toolkit to the /dev partition. It can determine which file-system the device uses, print mounted partitions and write device information to a report.txt. The investigator is given the option to enter the partition to be imaged. Once imaging is complete, the investigator can repeat process and image another partition or another device.

4. Remove Android Xtract – Once imaging has finished this program will remove programs and traces left over from Android Extract.

5. Swipe Unlock – This program disables the swipe lock and gives user access to the device.

6. AFLogical - This short program pushes and installs AFLogical and disables the screen-lock to allow the investigator to open the application and start the capture.

7. Remove AFLogical – Once AFLogical has finished this short script will remove AFLogical and Anti-Guard.

8. Android Live data Forensics – this program will install Lime and capture RAM.

9. Get Photos search device for photos and extract if found

10. Get Google data – Extract Browser history from device.

11. Com.android.providers Extract - Extract all data from com.android.providers.telephony, contacts, calendar, downloads, media, security, settings, applications and userdictionary

Android Forensics Tools (Cross-compiled for Arch ARM Linux)

1. Adb for arm
2. Android Xtract (Perl Script for imaging)
3. Antiguard.apk
4. Brute force python script
5. AflogicalOSE.apk
6. 70-android.rules (Device list)
7. Volatility (Is now included in Kali, so not needed)
8. Lime.

Raspberry Pi Operating system

This is an image of Kali Linux, with the above listed tools and scripts pre-installed.

Android P2P Forensics

1. Readme.txt – Describes the environment and tools needed to adb from Android to Android and to run Perl scripts from Android devices.
2. sl4a-r6.apk - (sl4a = Scripting Language for Android)
3. Android Perl Scripts – All of the Perl scripts, which have been altered to run from an Android Device.

References

Ableson F, Sen R. & King C.- *Android in Action*, published by Manning,
1.1 *The Android platform*
ISBN: 978-1-935182-72-6

ACPO (n.d). *Good Practice Guide for Computer-Based Electronic Evidence*. Retrieved March 7th, 2013 from;
http://www.7safe.com/electronic_evidence/ACPO_guidelines_computer_evidence.pdf

AFLogical – Retrieved April 14th from;
https://viaforensics.com/resources/tools/android-forensics-tool/

Alephzain (July 2013) *Framaroo*t. Retrieved May 5th, 2013 from;
http://forum.xda-developers.com/showthread.php?t=2130276

Andrew Hoog (2011) - *Android Forensics Investigation, Analysis, and Mobile Security for Google Android*, Pages 14, 218-228,268, 278-284
ISBN: 978-1-59749-651-3

Android Blog (June 2009). *The Android boot process from power on.* Retrieved January 25, 2013 from;
http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Android Developers (n.d), Dashboards (*June 2013), Platform Versions Retrieved June 10th, 2013 from;*http://developer.android.com/about/dashboards/index.html

Android Developers (n.d.) *Managing the Activity Lifecycle*Retrieved February 2, 2013 from;

http://developer.android.com/training/basics/activity-lifecycle/index.html

Android Developers (n.d.)*Storage Options.* Retrieved February 11, 2013 from;
http://developer.android.com/guide/topics/data/data-storage.html

Ballenthin W. (n.d.)*The Sleuthkit and Ext4*, Retrieved June 3rd, 2013 from;
http://www.williballenthin.com/ext4/

Cannon T. (Posted 8th November 2010) - *Android Reverse Engineering,* Retrieved 28th January 2013

from; http://thomascannon.net/projects/android-reversing/

Danchev D. (August 2010) *Researchers use smudge attack, identify Android passcodes 68 percent of the time,* Retrieved May5, 2013 from http://www.zdnet.com/blog/security/researchers-use-smudge-attack-identify-android-passcodes-68-percent-of-the-time/7165

Dent S. (March 2013) Galaxy *Note II vulnerability lets attackers (briefly) access home screen apps*. Retrieved March 28, 2013 from;
http://www.engadget.com/2013/03/04/galaxy-note-ii-homescreen-lock-vulnerability/

DFRWS 2011- *Forensics Challenge results*, retrieved 21st December 2012 from;

http://www.dfrws.org/2011/challenge/results.shtml

FAQoid (December 2012) - *Android Timeline and Versions* Retrieved January 7,2013

http://faqoid.com/advisor/android-versions.php

Forensic Blog (February 2012) *Cracking PIN and Password Locks on Android.* Retrieved May 21st, 2013 from; http://forensics.spreitzenbarth.de/2012/02/28/cracking-pin-and-password-locks-on-android/

Francis D'sa (June 2012) - *How to: Make your own USB OTG cable for an Android smartphone.* Retrieved June 13, 2013 from; http://tech2.in.com/how-to/accessories/how-to-make-your-own-usb-otg-cable-for-an-androd-smartphone/319982

Garfinkel S. (2011) - *Android Forensics*. Retrieved 1 December, 2012; http://simson.net/ref/2011/2011-07-12%20Android%20Forensics.pdf

Gargenta M (2011) - *Learning Android* published by O'Reilly, Chapter 1 Android Overview ISBN: 978-1-449-39050-1

Grundy B. (2004) - *The Law Enforcement and Forensic Examiner Introduction to Linux A Beginner's Guide.* Retrieved February 19th, 2013 from; http://www.rootsecure.net/content/downloads/pdf/forensic_guide_to_linux.pdf

GSM-Technology (n.d.) *JTAG Samsung i9100/i9100P (Galaxy S2),* Retrieved May 19th, 2013 from; http://www.gsm-technology.com/index.php/en_US,details,id_pr,8703,menu_mode,categories.html

Hale M. (February 2013) Volatility – *Android Memory Forensics*, retrieved June 7th, 2013 from https://code.google.com/p/volatility/wiki/AndroidMemoryForensics

Hashimi S. & Komatineni S(2009*) Pro Android*, published by O'Reilly, Chapter 1 *Introducing the Android Computing Platform*
ISBN: 978-1-4302-1597-4

Holson L. & Helft M. (August, 2008) T-*Mobile to Offer First Phone With Google Software.* Retrieved October 29, 2012; http://www.nytimes.com/2008/08/15/technology/15google.html

HtcDev,(n.d.) *Unlock Bootloader*, Retrieved February 2nd 2013 from;
http://www.htcdev.com/bootloader/

IOIO Wiki (February, 2013) - *IOIO Board Documentation*. Retrieved February 29th, 2013 from;
https://github.com/ytai/ioio/wiki
https://github.com/ytai/ioio/wiki/Getting-To-Know-The-Board

[Ivo Pooters (Posted 24th June 2012) - *Turning Android Inside Out*, Retrieved January 11th 2013 from;http://ebookbrowsee.com/d1t1-ivo-pooters-turning-android-inside-out-pdf-d393085365

Lessard J. & Kessler G *(2010) Android Forensics: Simplifying Cell Phone Examinations.* Retrieved January 11, 2013; http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=7480&context=ecuworks

Lime Forensics (March 2013) Retrieved March 14th, 2013 from;
https://lime-forensics.googlecode.com/files/LiME_Documentation_1.1.pdf

Manning C. (n.d*.) Building Yaffs into Linux.* Retrieved December 12th, 2012 from;

http://www.yaffs.net/preparing-build-linux-yaffs
http://www.yaffs.net/building-yaffs-linux
http://www.yaffs.net/test-yaffs-under-linux-using-virtualbox-and-vagrant

Markoff, J (November 2007). *I, Robot: The Man Behind the Google Phone*. The New York Times. Retrieved 2012-11-23; http://www.nytimes.com/2007/11/04/technology/04google.html
Memory Technology Devices (n.d.) *Nand Simulator.* Retrieved March 4th, 2013 from;

http://www.linux-mtd.infradead.org/faq/nand.html
Messmer E, (2012 October) *Getting forensics data off smartphones and tablet.* Retrieved January 5, 2013; http://www.networkworld.com/news/2012/101212-smartphone-forensics-263306.html

OHA, (2007) *Industry Leaders Announce Open Platform for Mobile Devices.* Retrieved December 7, 2012; http://www.openhandsetalliance.com/press_110507.html

Osborn K. (n.d.) *p2p-adb Framework*, Retrieved June 1st, 2013 from;

https://github.com/kosborn/p2p-adb

Parmar K. (NOVEMBER 2012) - *In Depth : Android Boot Sequence / Process*

http://www.kpbird.com/2012/11/in-depth-android-boot-sequence-process.html
Petazzoni T. 2013-06-07 The Buildroot user manual, Retrieved June 11th from;

http://buildroot.uclibc.org/downloads/manual/manual.pdf
Rizwan Ahmed and Rajiv V. Dharaskar- *Mobile Forensics: an Overview, Tools, Future trends and Challenges from Law Enforcement perspective.* http://www.iceg.net/2008/books/2/34_312-323.pd

Rosenberg D. (February 2013) - *Re-visiting the Exynos Memory Mapping Bug.* Retrieved March 21, 2013 from; http://blog.azimuthsecurity.com/2013/02/re-visiting-exynos-memory-mapping-bug.html

Samy M. (November 2010) *Introduction to Android App Development.* Retrieved January 19, 2013 from; http://mobileorchard.com/introduction-to-android-development/

Santoku Linux HowTo's, Retrieved January 15th, 2013 from; https://santoku-linux.com/howtos

Sogeti ESEC Lab (June 2012) *Forensics on Android phones and security measures,* Retrieved March 3, 2013 from; http://esec-lab.sogeti.com/post/Forensics-on-Android-phones-and-security-measures

Swartz K (September 2012) - HOWTO Brute Force Android Encryption on Santoku Linux. Retrieved June 13th, 2013 from; https://santoku-linux.com/howto/mobile-forensics/how-to-brute-force-android-encryption

Techgsm (n.d.) *XTC Clip HTC Android unlock,* Retrieved March 18, 2013 from; http://www.techgsm.com/XTC_Clip_HTC_Android_unlock,131169.html

The Open Source Android Forensics Toolkit, Retrieved March 19th, 2013 from; http://www.osaf-community.org/
The Unlockr (October 23, 2012) *How to root the HTC Wildfire S.* Retrieved April 7th from;

http://theunlockr.com/2012/10/23/how-to-root-the-htc-wildfire-s/
Vaas L. (April 2013) - *Viber flaw bypasses lock screen to give full access to Androids*. Retrieved April 3, 2013 from; http://nakedsecurity.sophos.com/2013/04/24/viber-flaw-bypasses-lock-screen-to-give-full-access-to-androids/

Wayne Jansen (Posted 2007, May) - Guidelines on Cell Phone Forensics. Retrieved March 11th, 2013 from; http://csrc.nist.gov/publications/nistpubs/800-101/SP800-101.pdf

Glossary of Terms and Abbreviations

- Root: Rooting means you have root access to your device—that is, it can run the sudo command, and has enhanced privileges allowing it to run apps like Wireless Tether or Set CPU. You can root either by installing the Superuser application—which many root processes include—or by flashing a custom ROM that has root access included.
- ROM: A ROM is a modified version of Android. It may contain extra features, a different look, speed enhancements, or even a version of Android that hasn't been released yet. To use a custom ROM, the Android device must be
- Flash: Flashing essentially means installing something on a device, whether it is a ROM, a kernel, or something else that comes in the form of a ZIP file. Sometimes the rooting process requires flashing ZIP file, sometimes it doesn't.
- Bootloader: Your bootloader is the lowest level of software on your phone, running all the code that's necessary to start up the operating system. Most bootloaders come locked, which prevents rooting the phone. Unlocking the bootloader doesn't root the phone directly, but it must be done in order to flash custom ROMs.
- Recovery: The recovery partition is the software on a phone for making backups, flash ROMs, and perform other system-level tasks. The default recoveries can't do much, but when a custom recovery is flashed, like ClockworkMod and after the bootloader is

unlocked it will give much more control over the device. This is often an integral part of the rooting process.

- ADB: ADB stands for Android Debug Bridge, and it's a command line tool for a computer that can communicate with an Android device that has connected to it by USB, Wi-Fi or Bluetooth. It's part of the Android Software Developers Kit (SDK). Many root tools use ADB, whether the user is typing the commands or not.
- S-OFF: HTC phones use a feature called Signature Verification in HBOOT, their bootloader. By default, the phone has S-ON, which means it blocks users from flashing radio images—the code that manages the data, Wi-Fi, and GPS connections. Switching the phone to S-OFF lets the user flash new radios. Rooting doesn't require S-OFF, but many rooting tools will give S-OFF in addition to root access.
- RUU and SBF: ROM Upgrade Utilities (for HTC phones) and System Boot Files (for Motorola phones) are files direct from the manufacturers, which change the software on the phone. RUU and SBF files are how the manufacturers deliver over-the-air upgrades, which are often post leaked by Android enthusiasts and hackers. They can also be used when downgrading a phone, if a rooting method isn't available for the newest software version yet. RUUs can be flashed directly from a HTC phone, but Motorola users will need a Windows program called RSD Lite to flash SBF files.
- JTAG - Joint Test Action Group (JTAG) is the common name for the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. It was initially devised by electronic engineers for testing printed circuit boards using boundary scan and is still widely used for this application, but today JTAG is also widely used for IC debug ports.
- USB OTG - USB On-The-Go is a specification that allows USB devices such as digital audio players or mobile phones to act as a host, allowing other USB devices such as flash drive, mouse, or keyboard to be attached to them. Unlike conventional USB systems, USB OTG systems can drop the hosting role and act as normal USB devices when attached to another host. This can be used to allow a mobile phone to act as host for a flash drive and read its contents.
- MTD - Memory technology devices (MTD) are a new type of device file in Linux for interacting with flash memory, similar to Flash Translation Layer. The MTD subsystem was created to provide an abstraction layer between the hardware-specific device drivers and higher-level applications. Although character and block device files already existed, their semantics don't map well to the way that flash memory devices operate.
- ACPO - The Association of Chief Police Officers (ACPO) brings together the expertise and experience of chief police officers from the United Kingdom, providing a professional forum to share ideas and best practice, co-ordinate resources and help deliver effective policing which keeps the public safe.
- Ext4 – Is the fourth extended filesystem for Linux, developed as the successor to ext3.
- YAFFS- Yet Another Flash File System is a robust log-structured file system that holds data integrity and high performance as a high priority. It is also designed to be portable and has been used on Linux, WinCE, pSOS, eCos, ThreadX, and various special-purpose Operating Systems.