Edith Cowan University
Research Online

Australian Digital Forensics Conference | Security Research Institute Conferences

2011

Forensic analysis of the android file system YAFFS2

Darren Quick
University of South Australia, Adelaide

Mohammed Alzaabi
Khalifa University of Science, Technology, and Research, Sharjah, UAE

Originally published in the Proceedings of the 9th Australian Digital Forensics Conference, Edith Cowan University, Perth Western Australia, 5th -7th December 2011
This Conference Proceeding is posted at Research Online.
http://ro.ecu.edu.au/adf/101

FORENSIC ANALYSIS OF THE ANDROID FILE SYSTEM YAFFS2

Darren Quick[1], Mohammed Alzaabi[2]
[1]University of South Australia, Adelaide, Australia
darren.quick@postgrads.unisa.edu.au

[2]Khalifa University of Science, Technology, and Research, Sharjah, UAE
mohammed.alzaabi@kustar.ac.ae

Abstract

The popularity of Android devices has resulted in a requirement for a process to extract and analyse data in a forensically sound manner. There is a wide range of devices which use the Android operating system, and hence a standard process for forensic extraction and analysis for all devices is not possible. Many devices use the Yet Another Flash File System (YAFFS), which introduces an additional layer of forensic requirements. Focussing on the internal storage of a Sony Ericsson Xperia x10i, a process to extract both logical and physical data from the internal NAND memory is possible after gaining super user access. Data was extracted in different formats by using a variety of software processes, such as SuperOneClick, dd, xRecovery, NANDdump, Yaffs2utils and Android Debug Bridge. Analysis of the extracts was then undertaken to determine the type of data available from the different extraction methods, which included Logical file extraction, Physical data with YAFFS spare information, and also without the YAFFS spare data. The analysis showed that the NANDdump has generated a bit-by-bit dump of the internal flash memory.

Keywords
Android operating system, YAFFS file system, Forensic analysis

INTRODUCTION

Smartphones are rapidly replacing computers in functionality and popularity as more and more people use them for day to day activities. The Android operating system is rapidly gaining a large market share and is being deployed on a wide range of devices, including smartphones and tablets. There is a great need to be able to extract and analyze data from these devices in a forensically sound manner. However, as these devices are so new, there is a lack of information as to how to achieve this.

Each device may have different requirements to access, extract, and analyze data. Research was undertaken using a Sony Ericsson Xperia x10i, to determine a process to perform forensic analysis for this device, and the file system, Yet Another Flash File System (YAFFS), to determine if there is a generic process that can be undertaken for Android and YAFFS devices.

The published research for the Android platform and forensic methodologies is minimal. The prevalence of Android devices, however, indicates that further and more advanced research is required in order to handle such devices properly, particularly when legal proceedings are involved.

One of the first works conducted in this area was done by Lessard and Kessler (Lessard & Kessler, 2010). The authors investigated an Android Smartphone by acquiring a logical and physical image of the device. They also used Cellebrite, a mobile forensic tool, to acquire information from the device and perform a comparison between these methods. The result of the analysis showed that the logical examination had resulted in the least fragmented files and easily viewable data. However, an issue with this research is the use of the 'dd' command to obtain a physical image of the device. As we will see later in this paper, the 'dd' command doesn't produce a complete physical image of the device.

Timothy Vidas et al. (Vidas, Zhang, & Christin, 2011) proposed a new methodology for forensic data collection for Android devices. The methodology relies heavily on the recovery partition of the device. The recovery partition contains a recovery image which allows the device to boot without loading the operating system. A custom recovery image has been built by the authors and flashed to the recovery partition in the Android device. The recovery image has been designed to support some functionality like dumping the flash memory, allowing the execution of the su command (which is a command used to gain root permissions in Linux-based platforms), and adding some custom transfer binaries. After flashing the custom recovery image, the device is rebooted into

the recovery mode. Using the adb tool (explained in the Overview of the Used Tools section), the data can be collected from the device and transferred to a computer.

ANDROID OVERVIEW

Since the introduction of mobile devices, and in particular Smartphones, the technology of these devices has been rapidly developing. These devices are becoming as functional as traditional computing devices such as Laptops and Notebooks. The Google Android platform is an open source platform designed for mobile devices such as Smartphones and Tablets. It includes an operating system that is based on Linux kernel 2.6, middleware to facilitate programming features, and a set of key applications. Android market share has recently increased, and according to a Nielsen (U.S. Smartphone Market: Who's the Most Wanted?, 2011) study conducted in March 2011, the Android Operating System had 50% of the total Smartphone new sales market share, followed by Apple iOS at 25%.

To forensically examine any digital device, it is essential to first identify the sources of digital information. With Android devices, there are four sources of interest: the internal storage, SD card, RAM, and the SIM card. In this research, our focus is on the internal storage.

In order to provide an advanced experience for Android developers, Android offers a persistent data storage using SQLite. Each application can store its relevant data in an SQLite database. Therefore, this will be a major place where an investigator can find information.

YAFFS (YET ANOTHER FLASH FILE SYSTEM)

The file system used in many Android devices is "Yet Another Flash File System" (YAFFS), which was designed to be a journaling file system for NAND Flash devices (Manning, 2002). Flash memory, as used in many portable electronic devices, is a non-volatile electrically erasable programmable read only memory (EEPROM). There are two types, NOR (Not-OR) and NAND (Not-AND), with the difference being that NOR is bit-addressable, and NAND is block addressable. NOR memory can be used like RAM, with single bits being addressed and applications can run directly from it. NAND cannot address individual bits, and must access blocks of memory through a controller (Myers, 2008). Flash memory does not write to create a '1', instead, a write operation changes a '1' to a '0'. An erase operation sets bits to '1'. For NOR memory, this can be done to one or more bits. NAND memory requires all bits in a block to be written, but can do this much faster than NOR memory (Myers, 2008).

YAFFS was designed for use with Linux, but is able to be ported to other operating systems (Manning, 2006). YAFFS2 is a more recent release which supports larger memory devices and also supports YAFFS1. NAND memory is divided into blocks and chunks. YAFFS1 chunks are 512-bytes and a 16-byte Spare or Out Of Band (OOB) area. YAFFS2 can have larger chunks, up to 2048-bytes with 64-bytes of Spare (Manning, 2006). Groups of chunks are combined to become an erase block, usually 64 chunks. A write to a chunk will clear bits, and an erase operation must be run across an entire block, which takes longer. Chunks can also be only written between one and four times between each erase, and there is a limit to the number of times an erase cycle can be done before the entire erase block burns out and cannot be used again. It was shown in Myers (2008) that this limit lies in the range of 10,000 to 100,000 times.

Chunks will consist of file data or an Object Header. An Object Header contains information relating to a file, such as the filename, size, creation date/time and Parent ID. Information about chunks is also stored in the Spare area. Hence a combination of the spare information and the Object Header information is needed to rebuild the file and folder structure. Figure 1 shows the structure of the Spare area, with each item a fixed length. Tables 1 and 2 list the structure of the YAFFS spare areas for YAFFS1 (16 bytes) and YAFFS2 (64 bytes) (Manning, 2006);

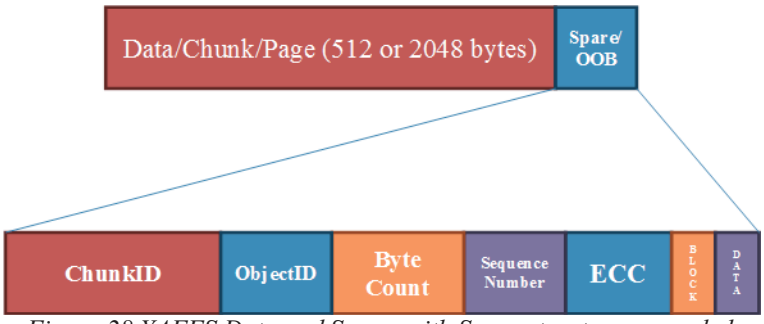

Figure 28 YAFFS Data and Spare, with Spare structure expanded

Table 11 YAFFS 1 Spare data structure

YAFFS 1 Spare (16 bytes) consists of;
8 bytes tags
6 bytes Error Correction Code (ECC)
1 byte block status (damaged)
1 byte data status (dirty)

Table 12 YAFFS 2 Spare data structure

	YAFFS 2 Spare (64 bytes) consists of;
bytes	
4	Chunk ID (20) (if 0 is a header (directory entry) if > 1 is data and position)
4	ObjectID (0 if unused)
2	nBytes, number of bytes used in the chunk, 0x 00 08 = 0x0800 = 2048 = full
4	Sequence number
3	ECC for tags
24	ECC for data
1	block status (damaged)
1	data status (dirty)

An Object Header contained within a chunk consists of the following information, as per Figure 2. Table 3 lists the structure of the Object Header (Manning, 2006);

YAFFS maintains a tree structure in RAM, and reads the entire memory at boot-up to build the structure. As NAND storage in devices gets larger, the boot time takes longer. Newer devices, such as those with dual-core processors, will move to hardware controlled wear levelling and a switch to the EXT4 file system rather than using YAFFS.

Each time a chunk is written, a sequence number in the spare area is incremented to identify the latest version. When data within a file changes, a new one is written with the updated data, and the original chunk is not changed. The sequence number is increased to reflect the later version of the data. The new chunk is written prior to the old one being discarded; hence there are potentially multiple older versions of data associated with a file, differentiated by the sequence number.

When a chunk is no longer valid, the data status byte in the spare area is marked as ‘dirty’. When all the chunks in an erase block are dirty, the block can be erased and reused. If free space is low, an erase block with a mix of allocated and dirty chunks can be used. The allocated chunks are copied to another area, and the entire block can then be erased.

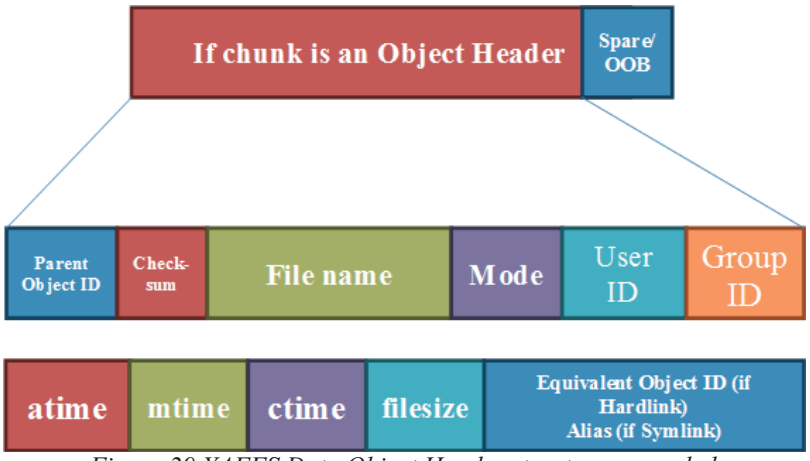

Figure 29 YAFFS Data Object Header structure expanded

Table 13 YAFFS Object Header data structure

Object Header (Directory Entry)	
bytes	
Integer:	Parent object ID
2	U16:sum was checksum of name, no longer used (left as FF FF)
255	charName
3	FF FF FF
4	u32: mode –protection – directory, file, symlink
4	Userid of owner
4	Group id of owner
4	atime – last access time
4	mtime – last modified time (data change)
4	ctime –last change time (Linux – file information change)
Integer:	filesize (files only) (if little endian, reverse to get hex to decimal) (zero if folder)
	equivalent object ID – for hardlinks
	alias for symlinks

COLLECTING DATA

Before starting the data collection phase, the Android device must first be made available for 'super user' access, commonly termed "rooting the device". Rooting an Android device refers to the ability of the user to gain root access and have root permissions. Actions such as accessing system files and changing file and directory permissions, along with many other actions, cannot be done without root access. The procedure of rooting Android device involves some modification to the device system files. One popular software application to root Android devices is *SuperOneClick* (SuperOneClick, 2010). In order to be able to access and image the flash memory of the device being analyzed, *SuperOneClick* was used to gain root access. The device used in this research is a Sony Ericsson Xperia X10i. It is usually undesirable to modify a seized device, however sometimes this is required to gain access to data. This research intends to study the YAFFS2 file system, and modifying the device is necessary to access this data. Prior to modifying a device, an examiner should research and understand the implications of the modification.

Another step prior to collecting data is to enable USB debugging mode in the device. This can be achieved by altering the application development setting in the device to enable USB debugging.

Overview of the Tools Used

A number of tools were used to collect data from the Sony Ericsson Xperia X10i. There are relatively few forensic tools that deal with the YAFFS2 file system. The tools that were used to conduct this research are:

- Android Debug Bridge (adb): adb is part of the Android Software Developers Kit (SDK). It provides a bridge between the Android device and the computer in order to allow a user to control and access the device's file system.
- dd command: dd is a Linux command that is used to copy storage media.
- NANDdump: NANDdump is a tool used to copy flash storage from devices. It is part of the Mtdutils utility (Mtdutils - Texas Instruments Embedded Processors Wiki, 2009) which has a collection of tools to interact with flash devices.
- xRecovery: xRecovery (xRecovery 0.3 xda-developers, 2010) is an open source custom recovery tool for Sony Ericsson Xperia series Smartphones. It allows a user to generate complete system backups of the device and also restore them.
- Yaffs2utils: Yaffs2utils (yaffs2utils, 2010) is a collection of tools that create and extract YAFFS2 image files. It includes: mkyaffs2 which creates YAFFS2 images from a set of files and directories, unyaffs2 which extracts a YAFFS2 image that was created by mkyaffs2, and unspare2 which extracts the spare space (Out-Of-Band) from a flash device.

Logical Acquisition

In order to obtain a logical image of an Android device, the adb tool can be used. The *pull* command in adb can copy the desired files from the device. The pull command can be specified as:

adb pull <remote> <local>

where <remote> is the path of the file/directory that will be copied from the Android device and <local> is the local path of a directory in the host machine where the pulled file/directory will be copied to.

In this process, a logical copy of the file system was obtained. From the Sony Xperia X10i, the acquired logical image had a size of 850 MB. In comparison, the internal storage of the Android device is 1 GB. After analysis of the logical image, it was determined that the pull command did not extract all files, omitting several system files.

Physical Acquisition

A physical image of an Android device is a bit-by-bit copy, in this case, targeting the internal flash memory. A physical image allows an investigator to conduct analysis on the entirety of the data, such as deleted data, which can have a great role in any investigation.

An important factor in this task is to validate the completeness of the physical image. In this research, different techniques and tools were used in order to obtain a physical image of the Android device and compare them. For example, comparison of the image of the data structure of the YAFFS2 files system and the size of the acquired image. The data structure in the file system can give an indication about the completeness of the image. Since the YAFFS2 file system can use a 2048 byte chunk size, the obtained image should also reflect that. In addition, the size of the obtained image should comply with the size of the device's flash memory.

The flash memory in Android devices is divided into number of partitions. These partitions are managed by the Memory Technology Device (MTD). In order to see the MTD partitions, the following command can be executed:

adb shell cat /proc/mtd

The result of executing this command for the Sony Ericsson Xperia X10i was:

dev:	size	erasesize	name
mtd0:	00440000	00020000	"appslog"
mtd1:	06f40000	00020000	"cache"
mtd2:	160a0000	00020000	"system"
mtd3:	1d100000	00020000	"userdata"

Each mtd partition has its own allocated space (size) as well as a name that indicates its content (name). The erazesize is 128K which is the erase block size.

Physical acquisition with dd command

As Android is based on the Linux kernel, the "dd" command can be used. The dd command can be issued for each mtd partition as follows:

dd if=/dev/mtd/mtd* of=/sdcard/mtd*.dd bs=4096

where * indicates the partition number. When analyzing a seized device, the resident SD card should be removed to be imaged separately, and another cleanly erased SD card should be used in its place to store the image files from the dd command.

After imaging all mtd partitions, each image should be validated to verify the data structure of the yaffs2 file system. The result of the validation indicated that the image files have only the chunk data without any spare data. Hence, the dd command does not produce a complete image of the flash memory.

This image cannot be realistically used to retrieve files. Due to the unavailability of the spare spaces, each chunk data cannot be mapped to the corresponding file object. The spare data structure, such as the object_id, is not available to facilitate this process.

Physical acquisition with xRecovery

xRecovery produces a backup of user and system data. After installing and running the application in the Android device, the backup option was selected. xRecovery generated three image files that correspond to; user, system, and cache data. These were stored on the SD card. The image files were retrieved from the SD card for further analysis and validation. The data structure of the image files was used to validate their completeness. In this example, each chunk data has its associated spare space.

Further analysis was conducted to understand how xRecovery works by looking at the source code of the application. The analysis showed that xRecovery uses "mkyaffs2" from the "yaffs2utils" tool in order to generate the image files. To prove this, "unyaffs2" was used to extract the files and directories from the image files, as unyaffs2 can work with an image generated by mkyaffs2. After applying unyaffs2 to the image files, the files and directories were successfully extracted. xRecovery produced image files of the logical data in the NAND memory. In addition, no deleted data was retrieved from these image files.

Physical acquisition with NANDdump

NANDdump appears to be the right tool to get a complete image of the flash memory. BusyBox (BusyBox, 2010) is a tool that compacts a set of Unix commands in a single executable file. The 1.18 version of BusyBox has introduced support for NANDdump. To image the mtd partitions of the device, the following command was used:

busybox nanddump <mtd-partition>

This enabled NANDdump to run on the Sony Ericsson Xperia x10i. The image files were validated against the data structure of the yaffs2 file system. The image files were divided into chunks which included spare spaces. The sizes of the partitions: apps log, cache, system, and userdata are 4.38 MB, 114 MB, 363 MB, and 479 MB respectively. Comparing these numbers to the allocated sizes of each partition mentioned earlier in this section, the sizes of the obtained images are accurate. Usually in digital forensics, an image is best verified by comparing its hash digest with the hash digest of the original data. With NAND memory, however, the hash digest is not stable and may produce different values if memory is imaged again (Hoog, 2011).

ANALYSIS OF YAFFS AND ANDROID

Android devices typically have memory card storage, such as Secure Digital (SD), MiniSD or MicroSD capability. The file and folder structure of the Sony Ericsson Xperia x10i included folders for audio, video, photographs, cache data, and application data. Information located on the SD card used in this research included references to the make and model of the device and the IMEI number. JPG pictures taken with the device contained EXIF data relating to the make and model. The information on an SD card can vary, so traditional forensic analysis methods are recommended for different devices.

Physical without the spare data

Analysis of the extracted data which did not have the spare information consisted of reviewing the hex data for object headers and file data. Data carving was undertaken with WinHEX, Scalpel and other carving software, and all produced similar results. Recovered data included Facebook thumbnail pictures, SQLite database files, XML, and web surfing remnants.

Physical with the spare data

To undertake analysis of the physical image which included the spare information, an operating system or software to interpret YAFFS is required. Native support for YAFFS is not included with the Linux kernel, however a kernel can be compiled to include YAFFS and then mounting of YAFFS images to view files and folders. The YAFFS file system is not supported by Microsoft Windows, or any current forensic analysis software, however it is understood support for YAFFS is being developed for AccessData's Forensic ToolKit, and also for The SleuthKit. For this research, the 'unyaffs' source code was compiled under Windows and Linux, and used to rebuild the folder structure of the xRecovery image files. Unyaffs rebuilds image files built using the 'mkyaffs' code, but does not work with images with the spare data created using NANDdump.

The structure of the spare area was outlined in Section 3, and analysis provides the size of the data in the chunk, the chunkID and ObjectID, the sequence of the data, and whether the data is current or marked as dirty. The image created using the xRecovery application included the spare area. Extracting the spare area to a spreadsheet was beneficial to compare the layout of the information for corresponding chunks.

Figure 3, displays spare information for nine object header entries and a data chunk for the "ansi" file. The object headers show "0xFFFF" as the number of bytes, and the ansi file has a value of "0xC9 05", which converts from little endian to 0x05C9, which is 1481 in decimal. The size of the data was confirmed with the extracted file, highlighted in Figure 4. The value in the spare relates to the number of bytes in the chunk allocated to data, not the total file size. A completely full 2048 byte chunk will have a spare size value of 0x800, or 0x00 80 little endian.

data/chunk	spare
	00100000010000000000000FFFF0000000000000000000000000000003
sys	00100000010100000000000FFFF0000000000000000000000000000003
tmp	00100000020100000000000FFFF0000000000000000000000000000003
usr	00100000030100000000000FFFF0000000000000000000000000000003
share	00100000040100000000000FFFF0000000000000000000000000000003
terminfo	00100000050100000000000FFFF0000000000000000000000000000003
a	00100000060100000000000FFFF0000000000000000000000000000003
ansi	00100000070100000000000FFFF0000000000000000000000000000001
<ansi file>	00100000070100000100000C9050000FFFFFFFFFFFFFFFF00000AFF90
local	00100000080100000000000FFFF0000000000000000000000000000003

Figure 30 Consecutive Spare data - listed in a column (truncated)

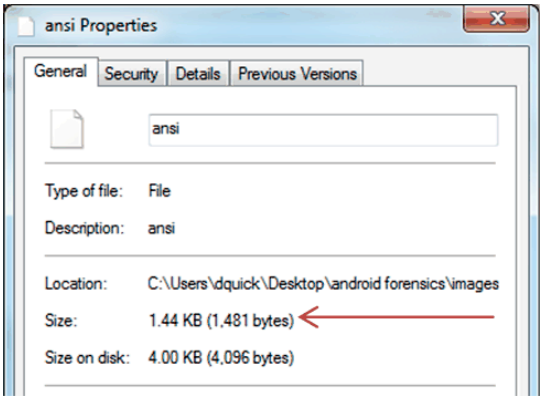

Figure 31 Arrow highlights the size of the data for the "<ansi file>" listing in Figure 3

Analysis of Object Header data using the structure outlined in Section 2, time information can be interpreted for the atime, mtime and ctime, which are stored as 32-bit Unix format. To convert the hex value to a date and time, the process is outlined as follows;

- Using an example value for an atime entry from an object header, 0x7A 4D 62 4D.
- When converted from little endian is 0x4D 62 4D 7A.
- This is 1298287994 in decimal, which is a 32-bit Unix date value.
- Which converts to “Mon, 21 February 2011 15:33:14 +0400”.

Conversion of the file size for the entire file is also possible from the object header;

- In Figure 5, the hex value for the filesize in the object header for is highlighted (0x7C 08 00 00), which is 0x087C when converted from little endian.
- This converts to 2172 decimal, hence the filesize is 2,172 bytes, which is confirmed in the properties of the extracted file, displayed in Figure 6.

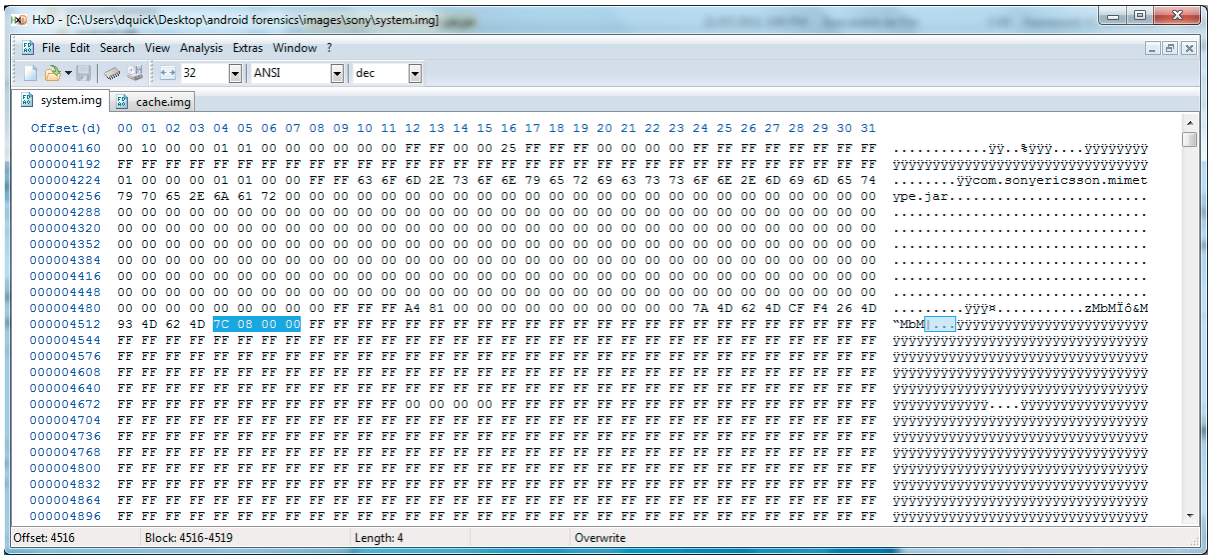

Figure 32 Highlight of file size data located in an Object Header

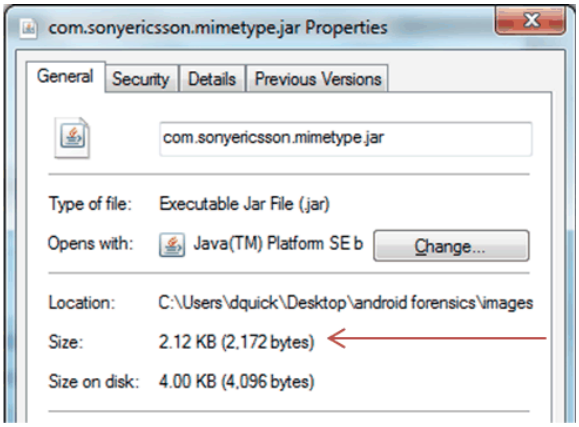

Figure 33 Highlight of the File size in comparison with the data from Figure 5

Logical files and folders

The extracted image created using xRecovery enabled the use of the ‘unyaffs’ program to build the file and folder structure of the NAND memory. xRecovery created three image files; “data”, “system” and “cache”. Located within the folder structure of the data.img file was a range of SQLite (.db) database files, XML files, and other data.

The information located within the SQLite database files included geocoded data, contacts, calls, SMS, browsing history, and Facebook information. A summary is included in Table 4. Data was located within XML files which included the IMEI number for the device. Stored passwords in hash and plaintext were also located for some applications.

Table 14 Examples of SQLite data located within the logical extract from a Sony Ericsson Xperia x10i

SQLite Browser (.db files)	Summary
GeoData	GPS cache position, latitude, longitude
Timezone	also stored in another file, "persist.sys.timezone" located in "data/property" in the format "Australia/Adelaide"
Bluetooth	Pairing ID numbers and names
Accounts.db	IMSI number
Usage logs	For a range of applications
Contacts	Names, phone numbers
Calls	Phone number and date/time information
SMS/MMS	Text, phone number and date/time
Dictionary	Typed words
Facebook	Friends, ID, jpg links, searches, photos, albums, updates, wall entries, email
Alarms	Time/date and text
Apps	Install times recorded for applications
Bookmarks	Internet browsing information
Searches	Internet browsing and device information
Calendar	Time/date and text for entries
Network connections	Details of networks, wireless
Downloads	Date/time and location
Meetings	Attendees
Email	Folders, sync times, message text
Pictures	Date added
Videos	Date added
Audio	Date added

CONCLUSION & FUTURE WORK

We have demonstrated a process for extracting logical and physical data from the internal NAND memory of the Sony Ericsson Xperia x10i. A number of different methods were use to extract data from the device, including logical files, physical data with YAFFS spare data, and physical data without the YAFFS spare data. The extraction process involved a range of tools, which are the dd command, xRecovery, and NANDdump. These extracts were then analyzed to determine the type and structure of data available. The analysis showed that NANDdump has generated a complete copy of the internal NAND memory.

Developing a tool that can read a YAFFS2 image and list the tree structure of the image's files and directories will be a significant contribution to the forensic computing realm. The tool should be able to interpret the spare spaces of the image file in order to detect file details. Given that the flash memory can store different version of the same file (or portion of a file) as a result of wear levelling techniques, the tool can also be designed to identify such data. Further work can also be done to capture an image of the RAM.

ACKNOWLEDGMENT

This research was undertaken when both authors were at the four-week inaugural IT security research winter school (jointly organised by stratsec and the University of South Australia).

REFERENCES

BusyBox. (2010, December). Retrieved October 12, 2011, from http://busybox.net/

Hoog, A. (2011, June). Geeks Guide to Digital Forensics. Retrieved from http://viaforensics.com/computer-forensics/google-tech-talk-geeks-guide-to-digital-forensics-june-2011.html

Lessard, J., & Kessler, G. C. (2010). Android Forensics: Simplifying Cell Phone Examinations. Small Scale Digital Device Forensics Journal, 4(1).

Manning, C. (2002, September). YAFFS: the NAND-specific flash file system - Introductory Article | YAFFS. Retrieved October 12, 2011, from http://www.yaffs.net/yaffs-nand-specific-flash-file-system-introductory-article

Manning, C. (2006, July). YAFFS Direct User Guide | YAFFS. Retrieved October 12, 2011, from http://www.yaffs.net/yaffs-direct-user-guide

Mtdutils - Texas Instruments Embedded Processors Wiki. (2009, March). Retrieved October 12, 2011, from http://processors.wiki.ti.com/index.php/Mtdutils

Myers, D. (2008, February). On the Use of NAND Flash Memory inHigh-Performance Relational Databases. Retrieved from http://people.csail.mit.edu/dsm/flash-thesis.pdf

SuperOneClick. (2010). Retrieved October 12, 2011, from http://forum.xda-developers.com/showthread.php?t=803682

U.S. Smartphone Market: Who's the Most Wanted? (2011, March). Retrieved October 12, 2011, from http://blog.nielsen.com/nielsenwire/?p=27418

Vidas, T., Zhang, C., & Christin, N. (2011). Toward a general collection methodology for Android devices. digital investigation, 8, S14–S24.

xRecovery 0.3 xda-developers. (2010, December). Retrieved October 12, 2011, from http://forum.xda-developers.com/showthread.php?t=859571

yaffs2utils. (2010, May). Retrieved October 12, 2011, from http://code.google.com/p/yaffs2utils/