# Digital Forensics Research Workshop Challenge 2011

## Technical Analysis

Author: Apurva Rustagi

7/24/2011

## Contents

## 1 Introduction

For the challenge, DFRWS provided physical data dumps from two Android phones. These dumps were captured using different methods.

The advantage of physical dump is that it allows us to find deleted data that is rather missed in logical extraction of data. However, the forensic examiner should be well-versed with intrinsic details of the underlying file system and additional software may be required to reconstruct the data. Android uses YAFFS2 as the underlying file system.

The challenge could have been tackled in two ways. The first way is to reconstruct the YAFFS2 file system from the raw dump provided. This requires high degree of familiarity with the file system internals and then development of tools to implement the reconstruction. Achieving this was not deemed practical in the provided time limit. Also, the metadata necessary for reconstruction of the file system was not captured in scenario 1.

The second way is to use file carving methods to carve out the SQLite databases from the data partition. Android stores most of the user data like SMS, emails, contacts, etc. in these databases. Hence, SQLite database structure was studied and some tools were developed to carve out information from these databases. This paper describes the concept behind implementation of those tools.

## 2 Mount points

One of the most interesting parts of any forensic analysis is the detailed examination of the file system, application and the data. Following is the list of mount points on Android phone found in scenario 1:

| Device | Mount point | File System | Options |
|---|---|---|---|
| tmpfs | /sqlite_stmt_journals | Tmpfs | rw,size=4096k 0 0 |
| /dev/block/mtdblock4 | /system | YAFFS2 | Ro 0 0 |
| /dev/block/mtdblock6 | /data | YAFFS2 | rw,nosuid,nodev 0 0 |
| /dev/block/mtdblock5 | /cache | YAFFS2 | rw,nosuid,nodev 0 0 |

The sqlite_stmt_journals file system in Android replaces the swap partition in conventional Linux systems. The activity on sqlite3 databases could use this partition to perform operations quickly in RAM. Since Android devices do not have swap space, RAM memory needs to be imaged for checking any remnants of important data.

The remaining partitions are all YAFFS2 (Yet Another Flash File System 2). The system partition is read only and remains unmodified by the end-user unless they gain root access.

The cache directory contains cached, unlinked and deleted files. This cache directory is used for Gmail attachment previews, downloads of files containing DRM, and downloads of applications from the Market place and OTA (Over the Air) system updates.

The most important directory from an investigation point of view is the /data directory and hence will be of main focus in the challenge. The agent, in scenario 2, did not collect the above data and hence it was no clear which image had the user data. However looking and comparing the size of the images, it was assumed that mtd8.dd in scenario 2 was the /data partition.

The /data directory has the structure illustrated in the following table.

| /data | |
|---|---|
| /anr | Debug information with timestamps from apps running in Dalvik Virtual Machine |
| /app | Application files (.apk) |
| /dalvik-cache | The .dex files (Dalvik Virtual Machine Executables) |
| /data | Subdirectory per application with associated data including preferences unstructured files and SQLite3 databases for structured data. |
| /misc | Non-application data, including information about dhcp, wifi, etc. |
| /property | Localization information including country, language, locale and time zone |
| /system | List of all applications and their permissions (packages.xml), checkin.db which lists many network, system and application events, syncmanager.db which lists results from various Gmail sync applications and several other files. |

The subdirectories in /data/data contain the data from the applications running on the phone.

| | |
|---|---|
| com.android.browser/ | Contains 7 directories including web icons, Google gears data, thumbnails, full cache with file and supporting database, cookies, bookmarks, searches and user/pass/form data in clear text |
| com.android.providers.calendar/ | Sqlite3 database with full calendar information |
| com.android.providers.downloads/ | Sqlite3 database with downloads for the system. Deleted records can be found with strings or by reverse engineering the format |
| com.android.providers.telephony/ | Sqlite3 data for all text messages |
| com.google.android.apps.maps/ | Google Maps search history and suggestions |
| com.google.android.providers.gmail/ | Gmail information including attachments, sender, receiver and full message |

## 3 YAFFS (Yet Another Flash File System) & OOB data

The memory for YAFFS file system is addressed in blocks and each block contains a set number of pages called chunks. For Android Devices, each block has 64 chunks and each chunk is 2KB. Hence the size of each block is 128KB. The block also includes a 64 byte Out of Band (OOB) area which is the spare area for storing various tags and metadata. When a block is allocated for writing it is assigned a sequence which starts at 1 and increments with each new block.

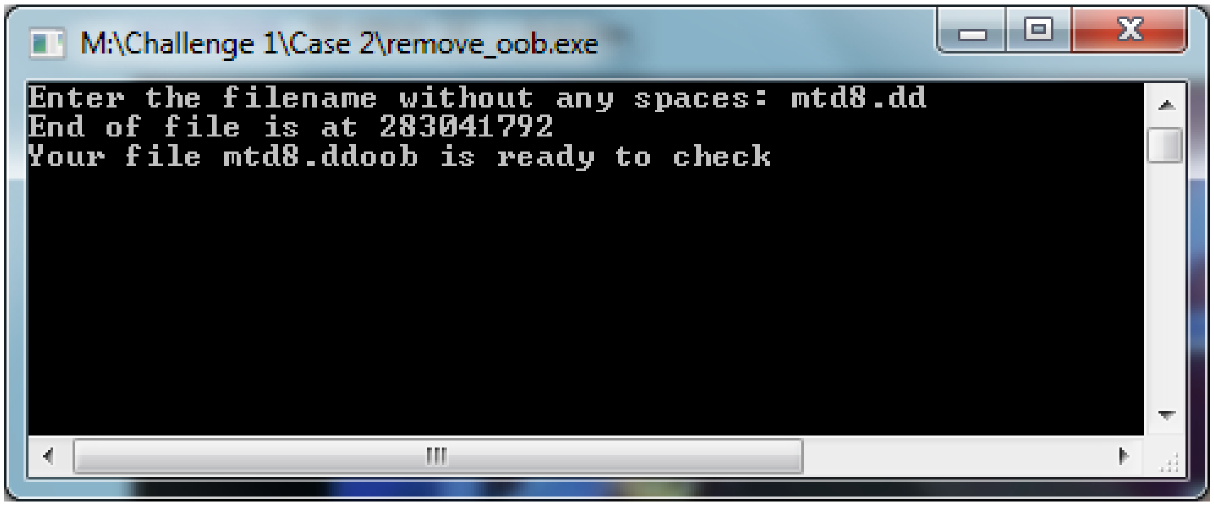
Data structures stored in YAFFS2 are referred to as Objects. These objects can be files, directories, symbolic links and hard links. Each chunk either stores a yaffs_ObjectHeader (object metadata) or data for the object. The yaffs_ObjectHeader tracks various information including the Object type, the parent object, a checksum of the name to speed up searching, the object name, permissions and ownership, MAC information and the size of the object if it is a file. In the 64 byte Out-of-band area, YAFFS2 stores critical information about the chunk but also shares the area with the Memory Technology Devices (MTD) subsystem. The critical YAFFS2 tags are:

- 1byte: Block State (0xFF if block is good, any other value for a bad block)
- 4byte: 32 bit chunk ID (0 indicates chunk is storing yaffs_ObjectHeader, else data)
- bytes: 32 bit object ID (similar to traditional UNIX node)
- 2 bytes: Number of data blocks in this chunk (all but final chunk would be fully
- allocated)
- 4bytes: Sequence number of this block
- bytes: ECC tags (in Android, handled by MTD)
- 12 bytes: ECC for data (in Android, handled by MTD)

If an object is changed, a new yaffs_ObjectHeader is written to flash since NAND memory can only be written once before erasing. The old data and headers still exist but are ignored in the file structure by examining the values of sequence number. Similarly, when a file is deleted in YAFFS, it is moved to a special, hidden "unlinked" or deleted directory. The file remains in this directory until all of the chunks in the file are erased. To achieve this, the file system tracks the number of chunks in the system for the file and when it reaches 0, the remnants of the file no longer exist. At that point, it will no longer track the object in the "unlinked" directory.

The OOB data forms the metadata of the filesystem and is necessary for reconstruction of the file system. However for carving we need the data area of the file system and since area of OOB is not separate from the data area, **remove_oob** was developed to extract only the data area of the file system.

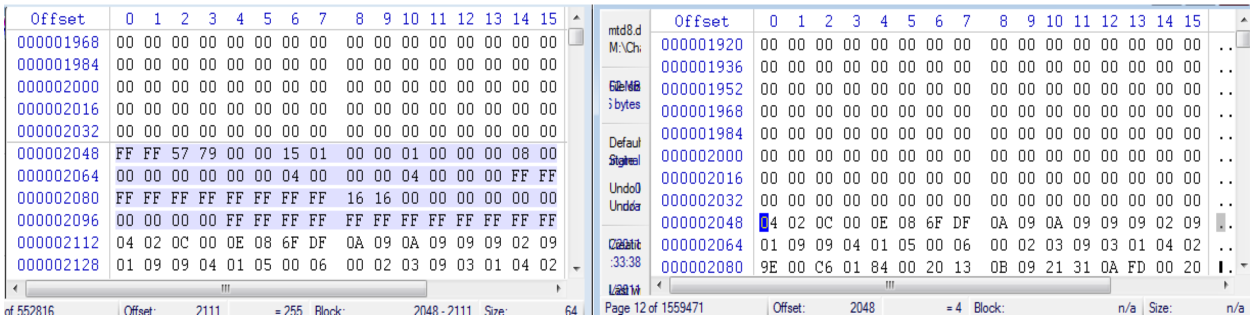
The program asks the filename containing the dump with oob data. It is recommended that you place the program in the same folder as the dump file. The program may take some time to process this file. The program will print the notification when it has finished processing.


M:\Challenge 1\Case 2\remove_oob.exe

```
Enter the filename without any spaces: mtd8.dd
End of file is at 283041792
Your file mtd8.ddoob is ready to check
```

The tool was used on mtd8.dd from scenario 2 and mtd8.ddoob was the resultant file. Following is the md5 hash of the resultant file:

| File | Size (bytes) | Hash |
|---|---|---|
| Mtd8.ddoob | 274466816 | D6762D2AD23289BD15D758FA01087EAB |



```
  Offset      0  1  2  3  4  5  6  7   8  9 10 11 12 13 14 15
000001968    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000001984    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000002000    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000002016    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000002032    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
000002048    FF FF 57 79 00 00 15 01  00 00 01 00 00 00 08 00
000002064    00 00 00 00 00 00 04 00  00 00 04 00 00 00 FF FF
000002080    FF FF FF FF FF FF FF FF  16 16 00 00 00 00 00 00
000002096    00 00 00 00 FF FF FF FF  FF FF FF FF FF FF FF FF
000002112    04 02 0C 00 0E 08 6F DF  0A 09 0A 09 09 09 02 09
000002128    01 09 09 04 01 05 00 06  00 02 03 09 03 01 04 02
of 552816   Offset:   2111   = 255  Block:   2048 - 2111  Size:   64
```

```
  Offset      0  1  2  3  4  5  6  7   8  9 10 11 12 13 14 15
000001920    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000001936    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000001952    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000001968    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000001984    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000002000    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000002016    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000002032    00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ..
000002048    04 02 0C 00 0E 08 6F DF  0A 09 0A 09 09 09 02 09  ..
000002064    01 09 09 04 01 05 00 06  00 02 03 09 03 01 04 02  ..
000002080    9E 00 C6 01 84 00 20 13  0B 09 21 31 0A FD 00 20  I.
Page 12 of 1559471   Offset:   2048   = 4  Block:   n/a  Size:   n/a
```

In the above image, the dump containing the raw data including OOB data is shown on left whereas the dump without the OOB data is on the right. The part marked in blue on the left is the OOB data.

## 4 SQLite Database files

The Android Operating System makes extensive use of SQLite files to store user data. Information such as contacts, call logs, browser history, user dictionary, etc. are all stored in SQLite databases. Customized tools were created for this challenge to carve out user data resident in these databases.

To understand the carving process, it is very important to understand the structure of SQLite database. The detailed explanation of the SQLite database is not in the scope of this document. The same can be found on the following links:

- http://www.sqlite.org/fileformat.html
- http://www.sqlite.org/fileformat2.html
- http://forensicsfromthesausagefactory.blogspot.com/2011/04/carving-sqlite-databases-from.html
- http://forensicsfromthesausagefactory.blogspot.com/2011/05/sqlite-pointer-maps-pages.html

To summarise, Android uses auto-vacuum capable SQLite databases. Such databases make use of Pointer Map pages. Pointer Maps are used to provide a lookup table to quickly determine what a pages parent page is. They only exist within auto-vacuum capable databases, which require the 32 bit integer value, read big endian, stored at byte offset 52 of the database header to be non-zero.

In auto-vacuum-capable SQLite databases page 2 of the database is always a Pointer Map page. Pointer Map pages store a 5 byte record relating to every page that follows the Pointer Map page. For example if we have an auto-vacuum-capable database that has 24 pages (each of 1024 bytes in size) in total, page 1 will contain the database header and the database schema and the next page, page 2, will be a Pointer Map page. This Pointer Map page will contain a 5 byte record for every one of the remaining 22 pages taking up 110 bytes of space within the page. The first 5 byte record begins at the very beginning of the Pointer Map page and therefore in a 1024 byte page a maximum of 204 (1024/5) records can be stored. If the database has more than 206 pages (when using a page size of 1024 bytes) page 207 would be an additional Pointer Map page that would contain records for the next 204 pages following this second Pointer Map page. Further additional Pointer Map pages can be added in the same way. Pointer Map pages do not store records relating to Pointer Map pages or page 1 of the database

Since all pages except page 1 and Pointer Map pages would have a 5 byte record, the number of records can be used to estimate the size of the database. The SQLite database header "SQLite format 3\000" is used to identify the start of the SQLite database file. **Sqlite_carver** program was developed to implement this concept and carve out the database files. Please note that this process can take several minutes to complete and hence it is advised to leave the program running. The program after carving below mentioned number of files does not end properly, but gets stuck in the while loop. This is because there is no particular End of File marker. The issue still needs to be debugged and solved. However, to verify that we cover all the SQLite headers, manual check was done to see if there was any SQLite header occurring after the last offset shown by the tool. There was no SQLite header remaining to be carved.

Following are the number of files carved from the data partition in each scenario:

| Scenario | File | No. of files carved |
|---|---|---|
| 1 | Mtdblock6.img | 1909 |
| 2 | Mtd8.ddoob | 1086 |

Among the large number of files carved, few files were properly carved and showed the data properly. However, many of the files in spite of having the data did not display in the SQLite browser. To solve this issue, db files were copied in a different folder.

## 4.1 Categorizing the SQLite files

To manage the large number of SQLite files, it was necessary to categorize them into different types. Each type of SQLite file like downloads.db, assets.db etc have a database schema which is stored in the first page of the SQLite database. This can be used as a basis for categorizing the files. However this is not a very certain idea.

**Category_case1** and **category_case2** programs were made to categorize the db files and place them in their respective folders. Here also, we saw some degree of success. All the files of type assets.db containing application information were placed in assets folder. File 1371.db in Case 1 shows information about twitter client installation. Other instances in assets folder show state of assets.db at various point of time. From these files we can see the state of twitter client from Download Pending ➔ Download Started ➔Installing ➔Installed.

However some of the files were not categorized and were placed in "misc" folder. Please note that here the executables must be in the same folder as the db files carved.

## 4.2 Viewing data in the B-Tree leaf nodes of SQLite databases

Since, not all information was shown in SQLite browser, "strings" tool was used to see information in each DB file carved. However there was too much of information to be managed and examined since this included schema of all SQLite db files which was not as important as the user data itself for the investigation.

SQLite databases store all the user data in B-Tree leaf pages and they start with 0x0D. Hence all the B-Tree leaf pages for each database were carved separately and then strings tool was used to examine data in each of them. The program name is **0Dpages.**

This data now formed the base of the investigation and gave many leads to the data we should be looking for, like SMS, email and pdf files.

## 5 Carving SMS records

SMS records were examined carefully and an attempt was made to identify the pattern. This would help in carving out them automatically rather doing it manually for each SMS. It was observed that SMS records started with a 10 digit phone number followed by a 6 byte UNIX time Stamp. The 6 byte UNIX time stamp, in this case started always with bytes 0x01 and 0x2F. Hence 12 byte header was identified to carve SMS for a particular contact. Since the phone numbers for SMS records were different in two scenarios, two versions of the tool were created. Following are the name of the files:

| Program name | Output file name | No of Records | Hash (MD5) |
|---|---|---|---|
| sms_carver_Case1 | Case1_sms | 4886 | E78F1A5086A3F17E62989EF2564F95C9 |
| sms_carver_Case2 | Case2_sms | 86 | ADFCA5B81564258FCADC026DC23F7D38 |

Using this technique, I was able to carve out SMS from data partition on Norby's phone between him, Mr. E and Taog. Sms_carver tool identified 4886 SMS records from Norby's phone and 86 records from Taog's phone. The tool dumped the records in delimited format which can be then analysed in Excel. This data was analysed to find the unqiue records and then sorted according to the time. Hence I was also able to find messages received during acquisition of the phone, which were unread. Following is the list of all the unread messages that were received during acquisition.

| Phone No. | Unix Time | Read | Type | Content |
|---|---|---|---|---|
| 4124393388 | 012fe023c165 | 0 | 1 | ksmsvzwsms://message/Service Started |
| 4124393388 | 012fe023d010 | 0 | 1 | ksmsvzwsms://message/May 8, 2011 5:41:16 PM EDT |
| 4124393388 | 012fe023e0e3 | 0 | 1 | ksmsFORWARDED SMS from 6245 at 20110508T045142America/New_York(0,127,-14400,1,1304844702) :shandra@cheerful.com (Thanks) Thanks for being so gracious last night |
| 4124393388 | 012fe023f0f2 | 0 | 1 | ksms Shandra |
| 4124393388 | 012fe023ff67 | 0 | 1 | ksmsvzwsms://message/Service Started |
| 4124393388 | 012fe0240eab | 0 | 1 | ksmsvzwsms://message/May 8, 2011 5:42:16 PM EDT |
| 4124393388 | 012fe0241e74 | 0 | 1 | ksmsvzwsms://message/Service Started |
| 4124393388 | 012fe0242cec | 0 | 1 | ksmsvzwsms://message/May 8, 2011 5:49:13 PM EDT |
| 4124393388 | 012fe0243bab | 0 | 1 | ksmsvzwsms://message/Service Started |
| 4124393388 | 012fe0244bfa | 0 | 1 | ksmsvzwsms://message/May 8, 2011 11:47:51 PM EDT |
| 4124393388 | 012fe0245a4e | 0 | 1 | ksmsvzwsms://message/Service Started |
| 4124393388 | 012fe02468e1 | 0 | 1 | ksmsvzwsms://message/May 9, 2011 12:03:09 AM EDT |

| Phone No. | Unix Time | Read | Type | Content |
|---|---|---|---|---|
| 4124393388 | 012fe0247734 | 0 | 1 | ksmsvzwsms://message/Service Started |
| 4124393388 | 012fe0248648 | 0 | 1 | ksmsvzwsms://message/May 9, 2011 12:44:18 AM EDT |
| 4124393388 | 012fe0249537 | 0 | 1 | ksmsFORWARDED SMS from 6245 at 20110510T032619America/New_York(2,129,-14400,1,1305012379) :shandra@cheerful.com (You around for lunch) Hey -- a few of us are go |
| 4124393388 | 012fe024a478 | 0 | 1 | ksmsing to that great Indian buffet for lunch today. You interested? |
| 4124393388 | 012fe024b47c | 0 | 1 | ksmsFORWARDED SMS from 6245 at 20110510T083450America/New_York(2,129,-14400,1,1305030890) :shandra@cheerful.com (Re: Sorry, still an Atlanta till) OK. Safe trav |
| 4124393388 | 012fe024c31f | 0 | 1 | ksmsels!  ----- Original Message ----- From: 4124393388@VTEXT.COM Sent: 05/10/11 09:31 AM To: shandra@cheerful.com |
| 4124393388 | 012fe024d1b2 | 0 | 1 | ksmsFORWARDED SMS from 4124393388 at 20110510T124241America/New_York(2,129,-14400,1,1305045761) :You have insufficient funds to send message. |
| 4124393388 | 012fe024e0a9 | 0 | 1 | ksmsFORWARDED SMS from 4124393388 at 20110510T124303America/New_York(2,129,-14400,1,1305045783) :You have insufficient funds to send message. |

## 6 Internet Searches

Various URLs indicating search on SwiftLogic and its employees were found during examination of B-Tree leaf pages retrieved from Norby's phone. Hence one more program was made to carve out the browsing history. It was observed that the browsing history records started with the signature 0x0105010001000000 followed by the title of the page, URL and six byte UNIX time stamp.

The **internet_history** program implements the carving and dumps the delimited records on the console. The records can then be copied into a text file and converted to columns in excel file.

## 7 Conclusion

The final sheets with consolidated data for each of the cases are **Case 1 Consolidated** and **Case 2 Consolidated**. The challenge was very interesting and it allowed me to increase my knowledge of SQLite databases and Android Operating System. I would like to thank DFRWS team for making such a great and interesting challenge.