International Journal of Hybrid Information Technology
Vol.8, No.7 (2015), pp.37-44
http://dx.doi.org/10.14257/ijhit.2015.8.7.04

Reconstructing Fragmented YAFFS2 Files for Forensic Analysis

Na Huang, Jingsha He, Bin Zhao, Gongzheng Liu and Xuejiao Wan,

School of Software Engineering
Beijing University of Technology
Beijing 100124, China

Abstract

Data recovery from captured intelligent mobile devices such as smartphones plays a significant role in digital forensic analysis. In this paper, we study the main characteristics of NAND flash and YAFFS2 file systems and explore the method for recovering YAFFS2 files for forensic analysis based on Tnode tree that can save a lot of time compared to other data recovery methods. For any broken file that has missing or broken data pages, we propose to reuse pages from previous versions of the current file based on the chunk IDs of the missing pages to replace and thus recover such pages. We will describe the replacement method with detailed steps and also perform some analysis to show that the proposed replacement approach can be feasible and effective in reconstructing YAFFS2 files.

Keywords: *Security; Digital Forensics; Data Recovery; YAFFS; Fragmentation*

1. Introduction

Nowadays, communication technologies raise various security issues while providing a lot of convenience to users. Since Android has become a major platform for developing communication technologies and applications, it has also become a major target for the acquisition and analysis of information that includes digital evidence. One of the great challenges in forensic analysis is the recovery of deleted damaged data that is possibly related to user behavior and could then be an important part of the digital evidence in the NAND flash memory which has been widely used in Android smart phones. And YAFFS2 is the file system that has been designed specifically for the NAND flash memory that Google has adopted as the official file system for Android smart phones and other types of intelligent devices.

There has been some research effort on data recovery. Luck and Stokes proposed an approach to recover files from handset memory dumps after studying the structure of the FAT file system [1], which afterwards was also applied to the Ext3 and Ext4 file systems. The method proposed in [4] performs data acquisition from Android smartphones regardless of versions and manufacturers. Spreitzenbarth et al. provided an overview of the YAFFS2 file system from the point of view of digital forensics and demonstrated how garbage collection and wear leveling techniques could affect the recoverability of deleted and modified files [6]. Sack et al. analyzed the structure of the Android system and formulated a forensic guide with the conclusion that all data stored on Android smart phones could be examined [7]. Sylve et al. discussed some of the challenges in performing Android memory acquisition and developed a new kernel module called dmd for memory dumping [8]. In the work, the kernel structure was analyzed through using newly developed volatility functionality and the results demonstrated the great potential that deep memory analysis could offer to digital forensic investigators. The conclusion was that if the spare area of flash memory pages doesn't exist or it is created from the unallocated area of the undamaged file system, reconstruction of the file system would not be possible [9].

ISSN: 1738-9968 IJHIT

To solve the problem of fragmentation, the authors also proposed some new analysis techniques for fragmented flash memory pages in smartphones. Wu et al. presented a recovery approach for SQLite history recorders from YAFFS2 that can correctly recover updated or deleted records in Android smart phones [11] through acquiring Android image in logical method and then recording the object ID, object type and chunk ID of each chunk sequentially to prepare for reversed-scanning. Based on the work in [5,11], Xu et al. further proposed a metadata-based method for recovering file traces that contain all the versions of files from YAFFS2 [12]. Although the above effort shows that it is possible to recovery files from NAND flash memory as long as the actual data still resides on the flash chip, none of the methods is shown to be able to recover broken fragmented files. The purpose of this paper is to further progress the previous effort by focusing on proposing methods for the recovery of YAFFS2 files.

2. NAND Flash and YAFFS2 File Systems

2.1 NAND Flash

Embedded systems traditionally have utilized the NOR Flash for nonvolatile memory. Many current designs are moving to NAND Flash to take advantage of its higher density and lower cost for high-performance applications. The 2GB NAND Flash device is organized as 2048 blocks with each block containing 64 pages and each page containing 2112 bytes which consists of a 2048-byte data area and a 64-byte spare area. The spare area is typically used for ECC, wear-leveling and other software overhead functions although physically it is the same as the rest of the page. NAND Flash is an electrically erasable and reprogrammable non-volatile flash memory with such advantages as low-power consumption, portable and low cost. It must be erased first before it can be rewritten, which is very different from the magnetic storage medium.

2.2 YAFFS2 File System

YAFFS (Yet another Flash File System) is specifically designed for NAND flash and YAFFS2 is the second version of YAFFS that has evolved from YAFFS1 to accommodate newer chips. There are two types of chunks in YAFFS2: data chunks and object header chunks [10]. Data chunks contain the actual file content while object header chunks contain file/directory metadata and descriptor information such as file size, object name and creation time [12]. Each chunk has Tags in the spare area that holds additional information such as the chunk ID, serial number, number of bytes and object ID. Every file that is stored on the NAND chips has a unique identity called the object IDs since files are regarded as objects by the file system. When a file is modified, the YAFFS2 file system will store the new data in empty pages and add a new object header [12]. When a file is deleted, the YAFFS2 file system will write a new object header at the end of the file to indicate that the file has been deleted. Only when most of the flash chip is used and there is data waiting to be written, will YAFFS2 invoke the garbage collection process [2].

Table 1 displays some information that is stored in the object header chunks and data chunks that is useful for data recovery according to YAFFS2 documentation [5] and our research, among which Yst_ctime indicates the time when the file was created, Yst_mtime indicates the time when the file was last modified and Yst_atime indicates the time when the file was last accessed which changes as the read operation is done. Listed in Table 1 is also the information in the object headers and tags. In a data chunk, object ID indicates to which object file the chunk belongs and value 0 means that the chunk is not part of any object. Chunk ID indicates the logical sequence of the chunk in the file to which it belongs and value 0 indicates that the chunk is a header chunk rather than a regular data chunk.

Table 1. Information in Object Header and Data Chunks

Content	Size(Bytes)	Comment
Parent_ Objectid	4	Parent directory
Name[yaffs_max_name_length+1]	256	File name
Yst_atime	4	Access time
Yst_mtime	4	Modified time
Yst_ctime	4	Create time
Filesize	4	Length of the file
isShrink	4	Whether the file is resized
Chunk ID	20	Sequence in the file
Object ID	18	Unique identifies of file

For YAFFS2, the content of each and every file is stored in chunks which are organized in the form of a Tnode tree for easy access. A Tnode tree provides a mapping to physical chunk address from file address [2]. Therefore, very object holds a Tnode tree. YAFFS2 Tnode tree is built with different pointers. Tnodes at the lowest level, i.e., Tnodes at level 0, has 16 physical chunk indexes that point to the chunk's logical location in memory [2]. Every Tnode has 8 pointers to point to the other nodes at the next level. We can use YAFFS_FindLevel0Tnode () to find the position of a physical chunk [2]. The tree must get updated whenever chunks get updated.

3. Recovering YAFFS2 Files Based on Tnode tree

When recovering files stored in a NAND flash, we need to scan the partitions of the flash first. For better performance, we choose the reversely-scanning algorithm that is illustrated in Algorithm 1 as follows. Let Blockn denote the block with sequence number n and Pagem denote the mth page in the block. We begin with the block with the largest sequence number and scan the whole block from the last chunk to the first one.

Algorithm 1:

Step 1: Scan $Block_n$;
Step 2: Scan $Page_m$ in $Block_n$. If $Page_m$ is the first one in the block, jump to Step 4;
Step 3: m=m-1, go to Step 2;
Step 4: If n=0, then end; else n=n-1, return to Step 1.

In YAFFS2, the strategy is designed to store new data in empty pages and to add a new object header when a file is modified or deleted. Fragmentation would then occur when files are modified. Traditional file recovering methods would scan the pages that belong to other files but are located between chunks of the current file and read their tags two or more times. The algorithm that we propose for recovering files and reconstructing the file system in YAFFS2 doesn't need to scan some fragmentation pages two or more times. Rather, it only scans each page in the NAND flash memory once, thus reducing the recovery time. The steps of our recovery algorithm are described in Algorithm 2 below:

Algorithm 2:

Step 1: Initialization n=1;

Step 2: Scan NAND flash using Algorithm 1; read the yaffs_tags from the spare area of each page in the NAND flash;

Step 3: Build the relevant Tnode tree for every file in which there are two different circumstances as follows:

(1) chunkID=0:

- If the yaffs_Object has been built in the RAM already, take the chunk with the latest Yst_atime;
- Else build a yaffs_Object according to its objectID;

(2) chunkID>0:

- If there exists an object header that has the same objectID, insert the present chunk into its yaffs_Object;
- Else if such an object header doesn't exist, build a yaffs_Object according to the objectID and insert the current chunk;
- Else if the chunk that shares the same chunkID has already been added into the relevant file's Tnode tree, determine which one is valid based on the Serial Number and abandon the invalid one;

Step 4: Once the construction of the Tnode tree of one file is complet, call yaffs_FindChunkInGroup() and yaffs_FindLevel0Tnode() to find the physical address of Level 0 pages and store them in File[n][i] where i= chunkID;

Step 5: Copy pages into the new area according to the physical address stored in File[n][i];

Step 6: If the current page is the last one in this NAND flash, then end the scanning; else n=n+1, go to Step 2.

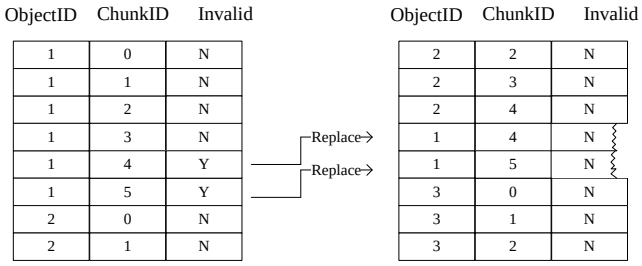


Figure 1. Recovery Techniques for Broken Files

4. Replace Method for Reconstructing Yaffs2 Files

When a file has been broken and some data pages or object headers could not be found, Algorithm 2 is no longer sufficient for the reconstruction of fragmented files. Consequently, we are not able to recover all the files in a NAND flash memory. Taking this situation into consideration, we propose a replacement method to attempt to reconstruct fragmented YAFFS2 files. The logic behind the idea is that, in YAFFS2, after a file is modified, the file system will store new data in empty pages and add a new object header to indicate that this file has a new version and the old pages are now made invalid [12]. We can thus use pages in previous versions of the same file according to the missing chunk IDs to locate and thus recover the missing chunks. However, the recovered file will be aligned to the previous versions instead of the latest one. We randomly picked 6 Android devices available and dumped the "/data" partition including the user data as

Image files using dd tools. All the devices we used in the experiments had been in use for at least 3 months and most of user data files thus have been modified. Consequently, the data pages are fragmented into many pieces as update, insert or delete operations were once performed. We used Frag-insight in our experiments which is a tool that is developed for analyzing the flash memory images (fragmented pages) and currently supports the YAFFS and Ext4 file system. We analyzed the 6 image files using Frag-insight and the results are showing in Table 2 where listed are the total size of each image file and the actual size of the storage space that all the files occupy in flash memory. Since each page consists of a 2048-byte data area, we can calculate the number of pages. All extra pages can be used to show that there are other history versions of the files stored in the NAND flash memory and can thus be used in our replace method for reconstructing files.

Table 2. Results of the Frag-Insight Analysis

	Image 1	Image 2	Image 3	Image 4	Image 5	Image 6
Total size	1.21GB	427MB	871MB	462.5MB	524.6MB	819MB
Valid file size	763.22MB	366MB	654.30MB	403MB	397MB	605.3MB
Number of pages	317194	109312	222976	118400	134297	209664
Number of valid pages	195384	93696	167501	103168	101632	154957
Extra pages	121810	15616	55474	15232	32665	54707

An illustration of some of the steps in the procedure in Figure 2 for recovering files is provided below:

Step 1: To determine whether the Tnode tree of a file is complete or not. It is complete when the number of pages in this Tnode tree is equal to Filesize in Object header.

Step 2: Put the ObjectID of a broken file in tmp[0].

Step 3: To determine whether the current page exists in Tnode tree. Scan the NAND Flash memory reversely and read the yaffs_tags from the spare area of each page. While ObjectID=tmp[0], match the ChunkID of this page with elements stored in tmp[i], $i \in (1,n)$. There are two different circumstances as follows:

(1) If the ChunkID of this page is equal to one of the elements in array tmp[], go on scanning;

(2) Else if the ChunkID of this page doesn't exist in array tmp[], insert it into the Tnode tree and store its physical address;

Step 4: Once the construction of the Tnode tree of this file is completed, copy all the pages into a new area. Clear array tmp[].

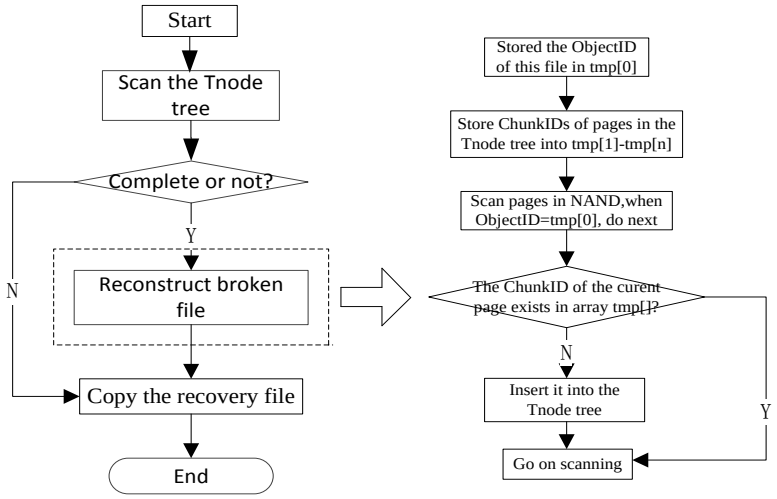


Figure 2. The Procedure for Recovering Missing Chunks

5. Analysis

We have performed some experiments to evaluate the proposed file reconstructing method to show its effectiveness and its superior performance over existing methods. We conducted this experiment over a VirtualBox set up with the Ubuntu operating system. We simulated a MTD in the random access memory RAM and created a simulation NAND flash using *nandsim* to mount YAFFS2 partitions. The storage capacity of NAND flash that we simulated is 64MB and the size of each page is 2KB. The following steps are taken to prepare the experimental environment.

1. Download the YAFFS2 source code package *yaffs2.tar.gz* and compile it.
2. Simulate a MTD in the random access memory RAM and create a simulation NAND flash using *nandsim* to mount YAFFS2 partitions.
3. Write files into the simulated NAND flash and modify them. Specific data are shown in Table 3.

When the reconstructing process is completed, data recovery and extraction are performed on the image files. Since our method uses pages in other versions of the file to replace the missing or broken pages, we would like to analyze how the changes in file size can affect the reconstructing results. Let Y denote the percentage of files that can be recovered and X denote the difference of file size between the different versions of the same file. Then, $X^T=(x_1, x_2,.....,x_m)$ and $Y^T=(y_1,y_2,.....,y_m)$. To estimate the parameters using the least squares method, Equation is as follows:

$$\begin{cases} \alpha = \dfrac{n\sum_{i=1}^{m} x_i y_i - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} y_i)}{n\sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{m} x_i)^2} \\ \beta = \bar{y} - \alpha \bar{x} \end{cases} \quad (1)$$

$$\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i, \bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i \quad (2)$$

Where the calculated results is $\alpha = -0.1123$, $\beta = 1.3830$. After analysis, the simulation curve is shown in Figure 3 which depicts the effect of recovery result caused by the change of file size.

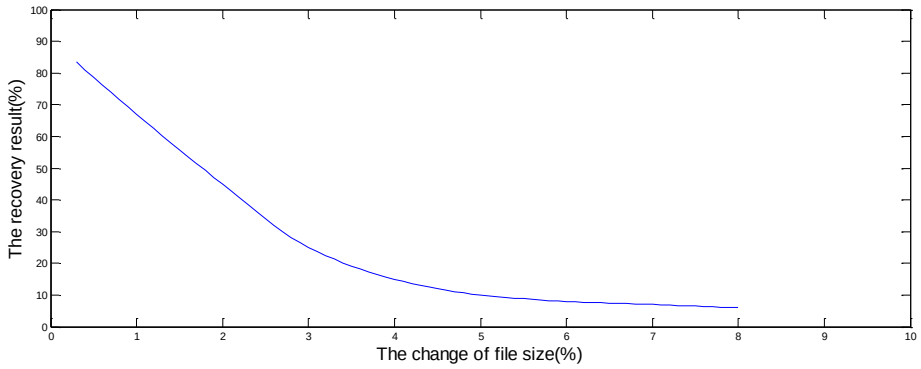


Figure 3. The Effect of File Size on the Recovery Result

6. Conclusion

In the field of digital forensic, many recovering techniques have been developed to extract user data as much as possible. To overcome the problem that high fragmentation occurred in storage devices, this paper demonstrates systematic reconstructing method only for YAFFS2 files. Our future research will focus on classifying the data pages of flash memory into different file types such as encrypted, compressed, image and video, and develop classification reconstructing techniques to recovery user data for each file type.

Acknowledgements

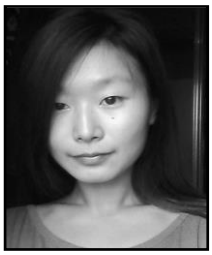
The work in the paper has been supported by National Natural Science Foundation of China (61272500) and Beijing Natural Science Foundation (4142008).

References

[1] James Luck, Mark Stokes. An Integrated Approach to Recovering Deleted Files from NAND Flash Data. Small Scale Digital Device Forensics Journal, 2(1), 2008:1-13.
[2] Charles Manning. How YAFFS Works. 2010. Available at: http://www.yaffs.net/documents/how-yaffs-works.
[3] Baiyi Huang. Data Recovery on Android Phones. M.S. Thesis, The George Washington University, May 2011.
[4] Andre Morum de L. Simao, Fabio Caus Sicoli, Laerte Peotta de Melo, Flavio Elias de Deus, Rafael Timoteo de Sousa Junior. Acquisition of Digital Evidence in Android Smartphones. In：Proceedings of the 9th Australian Digital Forensics Conference, Perth, 2011:116-124.
[5] Xue Yang, Ming Xu, Haiping Zhang. File Recovering from YAFFS2 based on Object Headers and Metadata. In：Proceedings of 4th International Conference on Graphic and Image Processing, Singapore, 2013: 876806 - 876806-8.
[6] Christian Zimmermann, Michael Spreitzenbarth, Sven Schmitt, Felix C. Freiling. Forensic Analysis of YAFFS2. Lecture Notes in Informatics P-195, 2012:59-70.
[7] Stefan Sack, Knut Kröger, Reiner Creutzburg. Overview of Potential Forensic Analysis of an Android Smartphone. Electronic Imaging, 2012：8304M-8304M-11.
[8] Joe Sylve, Andrew Case, Lodovico Marziale, Golden G. Richard. Acquisition and Analysis of Volatile Memory from Android Devices. Digital Investigation, 8(3-4), 2012:175-184.
[9] Jungheum Park, Hyunji Chung, Sangjin Lee. Forensic Analysis Techniques for Fragmented Flash Memory Pages in Smartphones. Digital Investigation, 9(2), 2012:109-118.

[10] Patrick Dibb, Mohammad Hammoudeh. Forensic Data Recovery from Android OS Devices: An Open Source Toolkit. In: Proceedings of 2013 European Intelligence and Security Informatics Conference, Uppsala, Sweden, 2013:226.
[11] Beibei Wu, Ming Xu, Haiping Zhang, Jian Xu, Yizhi Ren, Ning Zheng. A Recovery Approach for SQLite History Recorders from YAFFS2. In: Proceedings of Information & Communication Technology-EurAsia Conference 2013, Yogyakarta, Indonesia, 2013:295-299.
[12] Ming Xu, Xue Yang, Beibei Wu, Jun Yao, Haiping Zhang, Jian Xu, Ning Zheng. A Metadata-based Method for Recovering Files and File Traces from YAFFS2. Digital Investigation, 10(1), 2013:62–72.
[13] J. Katcher. PostMark: A New File system Benchmark. Technical Report TR3022, Network Appliance, 1997.www.netapp.com/tech_library/3022.html.

Authors



Na Huang, she is currently a M.S. student in the School of Software Engineering at Beijing University of Technology in China. Her research focuses on digital forensic.

He Jingsha, he received his B.S. degree from Xi'an Jiaotong University in Xi'an, China and his M.S. and Ph.D. degrees from the University of Maryland at College Park in USA. He is currently a professor in the School of Software Engineering at Beijing University of Technology in Beijing, China and an associate director in the Low Carbon Research Center at Beijing Development Area Co., Ltd. in Beijing, China. Professor He has published over 200 research papers in scholarly journals and international conferences and has received over 40 patents in the United States and in China. His main research interests include information security, network measurement, and wireless ad hoc, mesh and sensor network security.

