Digital Investigation 20 (2017) S83–S91

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2017 Europe — Proceedings of the Fourth Annual DFRWS Europe

AFEIC: Advanced forensic Ext4 inode carving

Andreas Dewald [a, *], Sabine Seufert [b]

[a] ERNW Research GmbH, Heidelberg, Germany
[b] Basys GmbH, Erlangen, Germany

ARTICLE INFO

Article history:
Received 26 January 2017
Accepted 26 January 2017

Keywords:
Digital forensics
Ext4 file system
Data recovery
Open source
Tool

ABSTRACT

In forensic computing, especially in the field of postmortem file system forensics, the reconstruction of lost or deleted files plays a major role. The techniques that can be applied to this end strongly depend on the specifics of the file system in question. Various file systems are already well-investigated, such as FAT16/32, NTFS for Microsoft Windows systems and Ext2/3 as the most relevant file system for Linux systems. There also exist tools, such as the famous Sleuthkit (Carrier), that provide file recovery features for those file systems by interpreting the file system internal data structures. In case of an Ext file system, the interpretation of the so-called superblock is essential to interpret the data. The Ext4 file system can mainly be analyzed with the tools and techniques that have been developed for its predecessor Ext3, because most principles and internal structures remained unchanged. However, a few innovations have been implemented that have to be considered for file recovery. In this paper, we investigate those changes with respect to forensic file recovery and develop a novel approach to identify files in an Ext4 file system even in cases where the superblock is corrupted or overwritten, e.g. because of a re-formatting of the volume. Our approach applies heuristic search patterns for utilizing methods of file carving and combines them with metadata analysis. We implemented our approach as a proof of concept and integrated it into the Sleuthkit framework.

Introduction

Data reconstruction plays an important role in the field of hard disk forensics (Casey, 2011) and it is specific to the used file system (Carrier, 2005). The Ext file system family is encounter as the standard file system on Linux and Android systems (Fairbanks et al., 2010). This paper illustrates an approach that enables reconstructing data without information from the superblock or the group descriptor table of the Ext4 file system.

Motivation

The Ext4 file system is a widely used file system, which is nowadays not only standard among Linux distributions, but is also used on mobile devices (Fairbanks et al., 2010). Ext4 and its predecessors save the metadata in the so-called superblock or the group descriptor table. Without these metadata structures it is difficult to interpret the file system correctly and to reconstruct the data. Of course, the remains the option of file carving, which however will not be able to recover file system metadata and on the other hand (besides specific techniques for some specific file types) is not able to cope with file fragmentation. The aim of this work is to reconstruct data without using metadata structures even in the case of overwriting or modifying the superblock through, for instance, overformatting. For previous Ext versions there are approaches which use the available contents of the metadata structures (Pomeranz). For example, on the Ext3 file system indirect block pointers in inodes – data structures where metadata is to be found on individual files – are saved on content data blocks. By dereferencing these block pointers, it is possible to reconstruct file contents. Referencing to the data contents on the Ext4 file system is handled differently, hence other techniques become necessary. Such a technique is introduced in this paper.

Contributions

This paper introduces an approach which was developed for finding and recovering files from an Ext4 file system. Moreover, the central metadata structures of the file system, such as the superblock and the group descriptor table, do not need to be available for our approach to work. Thus, a possible use case for our approach is

* Corresponding author.
E-mail address: research@andreasdewald.de (A. Dewald).

http://dx.doi.org/10.1016/j.diin.2017.01.003

when parts of the hard disk are overwritten or overformatted resulting in loss of the original metadata structures. In order to reconstruct files on the file system, we use search patterns, as it is known from file carving. However, instead of carving for specific file types, we carve for inodes. As there are no real magic bytes for an inode that can be used for simple carving, we build more complex patterns that identify valid inodes, what to the best of our knowledge has not been done so far. Carved potential inodes are then analyzed and the according files are recovered. By this means, our approach combines techniques form both, file carving, and metadata analysis. This way, not only the file content, but also the original file name and path can be reconstructed. We implemented our approach as a module for the Sleuthkit (Carrier), which is released open source along with this paper (Dewald and Seufert, 2017), where usage and configuration information is included, too.

Outline

After we discussed related work in the next section, in Section Ext4 file system novelties, we explain the novelties of Ext4 compared to Ext3. Section Methodology explains the methodology and implementation of our approach, which is thoroughly evaluated in Section Evaluation. We conclude our paper in Section Conclusion.

Related work

Brian Carrier (2005) describes in his book different partition and file systems. Carrier introduces different methods and tools to support the forensic analysis of the different file systems. The Sleuthkit (Carrier) is one of his developments, which provides various command line tools for digital forensics. On the one hand, we complement the work of Carrier, by highlighting the novelties in Ext4, and on the other hand, we implement a prototype of our introduced approach for Ext4 analysis as a plugin for the Sleuthkit Framework.

In his paper, Craiger (2005) describes digital forensic procedures for recovering data from Linux systems. He emphasizes the recovering of deleted and hidden files, data from volatile memory and files with modified extensions.

Fairbanks et al. (2010) compare the Ext4 file system with its predecessor from a forensic perspective, whose results we revisit in this paper. In another paper Fairbanks (2012) thoroughly describes the Ext4 file system and introduces the upgrade compared to Ext3. Moreover, the paper documents especially low-level features such as extents, HTrees and flex groups. Lee and Shon (2014) introduce procedures for recovering deleted files through metadata structures on Ext2 an Ext3 file systems and compare these with existing methods. Narváez (2007) describes a procedure to reconstruct files from an Ext3 file system using the journal.

The work of Pomeranz illustrates an approach to data recovery on Ext2 and Ext3 file systems that enables the recovery of user data by using indirect block pointers. The author exploits the fact that, typically, the first 48 KiB of a file content are not highly fragmented. Consequently, the first 12 block pointers are usually sequentially numbered, which similar to our approach applies some kind of specific carving. However, this particular search pattern cannot be applied to Ext4 file systems because normally (as we explain later) extent structures are used instead of indirect block pointers in order to reference file content. Nevertheless, the procedure used in our approach is similar to the one mentioned above.

Ext4 file system novelties

In this section, we summarize the relevant information about Ext4, as they are given in Ext4 disk layout (2016). The general layout of Ext4 is very similar to Ext3, but has changed in some ways that we want to focus on now. The Ext layout, in general, is based on sequential blocks of 1024, 2048 or 4096 bytes that are numbered and grouped together in block groups.

Each block group contains metadata that documents its inner structure. The general layout of all block groups is identical and is illustrated in Fig. 1. The superblock contains many essential metadata of the file system, such as the number and size of blocks, number of inodes and reserved blocks, for example. The following group descriptor table contains one group descriptor per block group in the file system and the block bitmap stores the free/used state of each block in the block group in a single bit each. Similarly, the inode bitmap stores the free/used state of each inode (entry) in the inode table. The rest of the block group consists of consecutive data blocks that are used to store data.

Inodes

In all Ext file systems, almost all file/directory metadata, such as timestamps, access rights, references to data blocks for example, are stored in the inode of the file (file names, for example, are not, although they are not always considered as metadata). Inodes are numbered, starting with inode number 1 and stored in the inode table of their respective block group.

In Ext4, for the sake of compatibility with prior versions, only few changes to the inode structure have been implemented. For example, some of the formerly unused space has been used to introduce new attributes, as shown in Table 1 (To recall the full original structure of Ext3 inodes, refer to Table A.9 in Appendix A).

To provide backwards compatibility, the concept of (single/double/triple) (in)direct block pointers to refer to content data blocks is still supported in Ext4. However, this concept is only used, when a old Ext3 file system is converted to Ext4. In all other cases, Ext4 uses an entirely new concept for data block references, which is called *Extents*, which are able to address more storage and allow for bigger files. Further, so called inline files and inline folders can be stored directly in the space for extended attributes. Fig. 2 illustrates exemplary how data blocks of a file are referenced by extents:

The inode of the example-file on the left side of the figure has 60 bytes to store its extent structure. The size of one extent entry is 12 bytes, thus there can be 5 extent entries stored directly. Each extent structure starts with an extent header with a size of 12 bytes as well. For completeness, the detailed structure of a extent header is shown in Table A.6 in Appendix A. The extent header is followed by extent

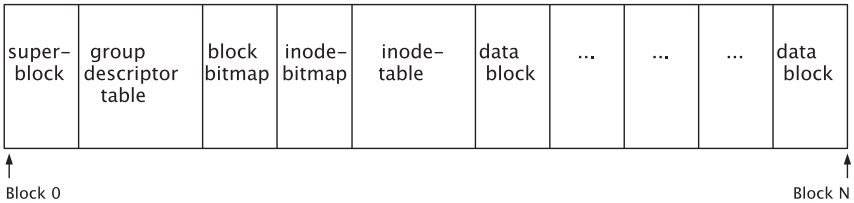

Fig. 1. General block group layout.