available at www.sciencedirect.com

ScienceDirect

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

Digital Investigation

Using the HFS+ journal for deleted file recovery

Aaron Burghardt[*], *Adam J. Feldman*

Booz Allen Hamilton, Herndon, VA 20171, United States

Keywords:
Mac OS X
HFS+
Journal
Deleted
File
Recovery

A B S T R A C T

This paper describes research and analysis that were performed to identify a robust and accurate method for identifying and extracting the residual contents of deleted files stored within an HFS+ file system. A survey performed during 2005 of existing tools and techniques for HFS+ deleted file recovery reinforced the need for newer, more accurate techniques.

Our research and analysis were based on the premise that a transactional history of file I/O operations is maintained in a Journal on HFS+ file systems, and that this history could be used to reconstruct recent deletions of active files from the file system. Such an approach offered a distinct advantage over other current techniques, including recovery of free/unallocated blocks and ‘‘file carving’’ techniques. If the journal entries contained or referenced file attributes such as the extents that specify which file system blocks were occupied by each file, then a much more accurate identification and recovery of deleted file data would be possible.

1. Background

Applications for recovering deleted files on Mac OS HFS and HFS+ file systems historically have had limited success compared to recovery tools for other common file systems; the difference is a consequence of HFS's use of B-tree data structures to store metadata that describes the name, block allocation, and other file attributes. When a file is deleted, the B-tree is immediately updated to maintain consistency, which overwrites the file metadata.

With the release of Mac OS X v10.2 in August 2002, Apple enhanced HFS+ by adding metadata journaling, which groups metadata changes into a transactional block. The journaling can be manually enabled or disabled on-the-fly by the user.[1] In version 10.2, journaling was disabled by default. Mac OS X v10.3 was released in October 2003, and it enables the journal by default. Therefore, the recovery technique described here is the most applicable on systems with v10.3 or later installed and on volumes formatted by v10.3 and later systems.

1.1. *The HFS+ file system*[2]

The major components of the HFS+ file system are:

- **Volume header** – contains file system attributes, such as the version and the allocation block size, and information to locate the metadata files.
- **Allocation file** – tracks the usage status of allocation blocks.
- **Catalog file** – contains the majority of file and folder metadata.
- **Extents overflow file** – contains additional extents records for files composed of more fragments than can be recorded in the catalog file.

[*] *Corresponding author.*
E-mail address: burghardt_aaron@bah.com (A. Burghardt).
[1] With either a command line tool diskutil or the *Disk Utility* application.
[2] A full description of the file system format is provided by Apple in Technote 1150.

1742-2876/$ – see front matter © 2008 Digital Forensic Research Workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2008.05.013

- **Attributes file** – extensible metadata; it is used for features such as access control lists and Time Machine.
- **Journal file**

The catalog, extents overflow, and attributes files are all instances of a B-tree storage system (Sedgewick, 1990). Only the catalog file is considered in detail here.

1.1.1. *The catalog file*

The catalog file stores catalog file records and catalog folder records, which are the primary store of file and folder metadata, respectively. These records contain:

- Catalog node ID (CNID, a unique unsigned 4-byte integer).
- Timestamps.
- Owner ID.
- Group ID.
- Unix permissions.
- Finder and miscellaneous properties.
- Eight extents records.

An extent record describes the starting block and length in blocks of contiguous blocks that are part of a single fork of a single file (i.e., a single fragment of that file). Thus, the eight extent records in a catalog file record identify the first eight fragments of the file. Additional extents per file are stored in the extents overflow file.[3]

Keys for catalog file and folder records are derived from CNIDs and file names. Each key contains the CNID of the *parent* folder and the name of the file or folder. Using the parent CNID keeps the contents of a folder grouped together in the B-tree nodes.[4]

The catalog file also contains thread records to improve performance when retrieving a file by CNID. A thread record key contains the CNID of the file or folder, but no file name. The data in a thread record is a copy of the key for the file or folder record. Thread records are useful when constructing a path to a file system object; given the key for a file or folder record, the most efficient way to find the key of the parent folder is to retrieve the thread record for the parent CNID. By repeating this process recursively until the root is reached, the full path to the object (relative to the file system's mount point) is obtained.

1.2. *The difficulty of deleted file recovery*

The organization of catalog file data implies that accurate recovery of deleted files can be achieved if the file record and its key can be found. Some existing COTS tools take this approach and employ a scan of the catalog file for deleted file records as the first step. The results of these utilities are often limited to one or two files or no files at all.

This approach is ineffective because the catalog file is not just an index of another data structure; the indexed data is stored within the catalog file. When a file is deleted, the B-tree must be updated to maintain consistency, which may overwrite the deleted file record. In HFS, these updates were performed in a way that occasionally did not erase the record in the B-tree node corresponding to a deleted file. In HFS+, B-tree nodes appear to be updated as a logical unit. In our research, we found that the slack/unused portions of B-tree nodes were consistently filled with 0x00.

1.3. *The journal's role*

An individual update to the file system, from the user's perspective, may result in several related disk updates. Creating a file, for example, may trigger the following changes:

- File and thread records inserted into one catalog file node (which may cascade into several nodes requiring updates).
- The Volume Bitmap file is updated to reflect that the file's content blocks are now in use.
- Records are inserted in the extents overflow if the file is highly fragmented.
- The attributes file is updated if additional attributes are applied.
- The volume header is updated to reflect activity.

All of these updates must be completed or the file system will be corrupted. A power outage or premature removal of an external disk or flash drive is an example where the file system may be interrupted while critical updates are in progress. Journaling was added to HFS+ to address this problem. The steps in a transaction are:

1. Start the transaction by writing a copy all pending file system metadata changes to the journal file.
2. Flush the journal file to disk.
3. Record the presence of the transaction in the journal header.
4. Perform the changes on the actual file system metadata files.
5. Mark the transaction in the journal as completed by updating the journal header.[5]

When a file system is mounted, HFS+ checks the journal for uncommitted transactions. If the transaction did not reach step 3 above, then the changes are lost, but the file system is still consistent. If step 3 completed successfully but step 5 did not, then the entire transaction is replayed, ensuring file system consistency.

1.4. *The journal file*

The journal file is allocated as a contiguous set of blocks on the file system and is never moved or resized. The journal file is implemented as a circular buffer of a fixed size. The beginning of the journal file is the journal header; the remainder of the file is the circular journal buffer. Journaled data is continuously

[3] In practice, only 1–2% of the files of a typical Mac OS X volume will have records in the extents overflow file (Singh, 2006; Singh).

[4] This is a performance optimization for common user tasks, such as displaying or searching the contents of a folder.

[5] There is no need to flush the journal header to disk at this point; disconnecting the disk and leaving the file system dirty will result in a replay of the transaction on the next mount, but that is not incorrect, only redundant.