Chapter 16

FILE SYSTEM JOURNAL FORENSICS

Christopher Swenson, Raquel Phillips and Sujeet Shenoi

Abstract Journaling is a relatively new feature of modern file systems that is not yet exploited by most digital forensic tools. A file system journal caches data to be written to the file system to ensure that it is not lost in the event of a power loss or system malfunction. Analysis of journal data can identify which files were overwritten recently. Indeed, under the right circumstances, analyzing a file system journal can reveal deleted files and previous versions of files without having to review the hex dump of a drive. This paper discusses data recovery from ReiserFS and ext3, two popular journaled file systems. It also describes a Java-based tool for analyzing ext3 file system journals and recovering data pertaining to overwritten and deleted files.

Keywords: File system forensics, journaling, ReiserFS, ext3

1. Introduction

Traditional computer forensics involves acquiring and analyzing file system images. Most forensic tools exploit file system features to obtain evidence. For example, the tools may find hidden or deleted data in FAT, ext2 and NTFS file systems by examining the slack space and free space, or by searching through the file system tree itself [3, 8].

Journaling is an advanced file system integrity feature [6, 10, 13] that is not exploited by most digital forensic tools. This feature is employed in virtually all modern file systems, including NTFS (Windows NT/2000/XP), HFSJ (Mac OS X), ext3 (Linux) and ReiserFS (Linux).

A file system journal works by caching some or all of the data writes in a reserved portion of the disk before they are committed to the file system. In the event of an unexpected power loss, malfunction or other anomaly, the journal could be replayed to complete any unfinished writes, preventing file system corruption due to incomplete write operations. This also means that previous file writes are stored for lim-

Please use the following format when citing this chapter:

Swenson, C., Phillips, R., Shenoi, S., 2007, in IFIP International Federation for Information Processing, Volume 242, Advances in Digital Forensics III; eds. P. Craiger and S Shenoi;(Boston: Springer), pp. 231-244.

ited periods of time in the journal, i.e., outside the normal file system. Therefore, even if a file is overwritten or securely deleted, it may be possible to recover the old contents by analyzing the journal. Indeed, under the right circumstances, analyzing a file system journal can reveal deleted files and previous versions of files without having to review the hex dump of the entire drive.

This paper focuses on file system journal forensics, with particular attention to the Reiser [9] and ext3 [12] journaled file systems. The next two sections describe the structure and organization of the Reiser (v. 3) and ext3 file systems, including their journaling features. Section 4 discusses how file system journals may be analyzed to recover data about overwritten and deleted files; it also describes a Java-based tool for ext3 journal data recovery and analysis. Section 5 highlights the experimental results obtained when searching ReiserFS and ext3 journals for information about overwritten and deleted files. The final section presents some concluding remarks.

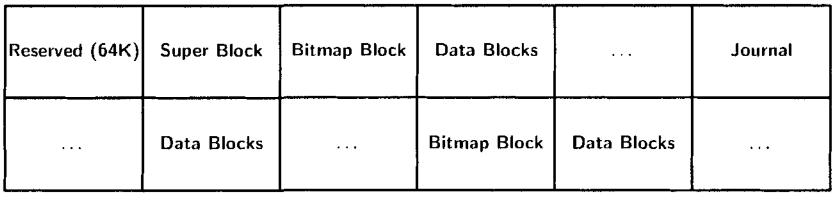


Figure 1. ReiserFS block structure.

2. Reiser File System

This section describes the structure of the Reiser File System (ReiserFS), including its journaling feature [1, 9].

2.1 ReiserFS Structure

ReiserFS has a block structure with fixed-size blocks (usually 4,096 bytes). The ReiserFS block structure is shown in Figure 1.

The superblock is the first block of the ReiserFS structure. The structure of a superblock is presented in Table 1. Unlike some other file systems (e.g., ext2), ReiserFS has only one copy of the superblock.

ReiserFS bitmap blocks are special blocks that identify used and unused blocks. Each bit in a bitmap block acts as a "used bit" for a single block in the file system.

Table 1. Superblock structure 3.6.

Name	Bytes	Description
Block Count	4	Number of blocks in file system
Free Blocks	4	Number of unallocated blocks
Root Block	4	Location of the root block
Journal Information	28	Various aspects of the journal
Block Size	2	File system block size
Object ID Max. Size	2	Maximum size of the OIDs
Object ID Current Size	2	Current size of the OIDs
State	2	Whether the partition is clean
Magic String	12	`ReIsEr2Fs`
Hash Function	4	File name hash function
Tree Height	2	Height of file system B-tree
Bitmap Number	2	Number of bitmap blocks
Version	2	Version of the superblock
Reserved	2	Reserved
Inode Generation	4	Rebalancing count

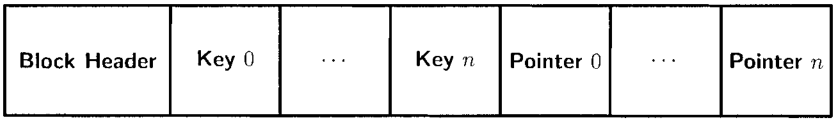


Figure 2. ReiserFS internal block contents.

ReiserFS has three types of data blocks: (i) unformatted data blocks, (ii) internal blocks, and (iii) leaf blocks. An unformatted data block contains raw data that corresponds to the contents of indirect files. An internal block contains pointers to data in the file system B-tree [4]. Figure 2 shows the contents of an internal block (internal B-tree node). A leaf block corresponds to the end node of the file system tree. It contains statistics, information about directories and possibly data itself.

2.2 ReiserFS Journal

ReiserFS in Linux has a fixed-size journal consisting of 8,192 4,096-byte blocks, plus a 4,096-byte header block, corresponding to a total size of approximately 32 MB. Note that the journal is only supported for 4,096-byte blocks. However, this is not a problem, as other block sizes are never used.

The header block uses the first 12 bytes to keep track of which block was last flushed and where to begin the next flush, as well as to mount information. The remaining 4,084 bytes are reserved.

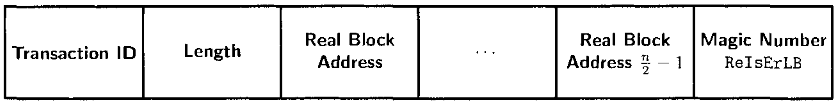


Figure 3. Transaction description block for ReiserFS.

The remainder of the journal comprises a series of transactions in a circular queue. Each transaction begins with a description block, identifying the transaction and providing half of a map that identifies where to put the data blocks (Figure 3). This is followed by the actual data blocks to be written.

A transaction is terminated by a final commit block that contains the remaining portion of the block map and a digest of the transaction. The structure is identical to the description block, except that the magic number is replaced by a 16-byte digest.

3. Ext3 File System

The ext3 file system extends the earlier ext2 file system [2] by adding a journaling feature [11, 12]. Ext3 has rapidly become the most popular journaled file system in Linux, and is usually the default choice for new installations [5]. This section describes the structure and journaling features of the ext3 file system.

3.1 Ext3 Structure

Ext3, like ReiserFS, is based on a block structure. The block size is the same throughout a single file system; depending on the file system size, however, the default value is either 1,024 or 4,096 bytes. The first 1,024 bytes of an ext3 file system are always reserved. If the file system contains the boot kernel, these bytes will hold boot information.

The ext3 superblock stores general file system information, e.g., name, block size and the last time it was mounted. An ext3 file system has one primary superblock, although backup copies may also be stored throughout the file system. The primary copy begins at the 1,024th byte. Table 2 provides details about the ext3 superblock structure.

An ext3 file system maintains file metadata in data structures called inodes. Inodes are stored in special blocks called inode tables. The inodes are 256 bytes long by default.

Table 2. Ext3 superblock structure.

Name	Bytes	Description
Inode Count	4	Total number of inodes in file system
Block Count	4	Total number of blocks in file system
Blocks Reserved	4	Reserved block count to prevent overfill
Free Block Count	4	Number of unallocated blocks
Free Inode Count	4	Number of unallocated inodes
Group 0	4	Block where first block group starts
Block Size	4	Left shifts of 1024 to obtain block size
Fragment Size	4	Left shifts of 1024 to obtain fragment size
Blocks per Block Grp.	4	Blocks in a typical block group
Fragments per Block Grp.	4	Fragment count in a typical block group
Inodes per Block Grp.	4	Inodes in a typical block group
Last Mount Time	4	Seconds from epoch to last mount time
Last Written Time	4	Seconds from epoch to last write time
Mount Information	4	Total and max. mounts of file system
Signature	2	`0xEF53`
File System State	2	Clean, error, recovering orphan inodes
Error Handling Method	2	Continue, remount as read only, or panic
Minor Version	2	Original or dynamic
Consistency Check	8	Last performed, interval
Creator OS	4	Linux, FreeBSD, etc.
Major Version	4	Original or dynamic
Reserved Block UID/GID	4	UID/GID that can use reserved blocks
First Inode	4	First non-reserved inode in file system
Inode Size	2	Size of inode in bytes
Block Grp. Loc. of Copy	2	If backup copy, group of copy
Feature Flags	12	Features of the file system
File System ID	16	UUID of file system
Volume Name	16	OS's name for the volume
Other Misc. Information	72	Misc.
Journal Information	24	UUID, metadata inode, device
Orphan Inodes	4	Head of orphan inode list
Unused	788	Unused bytes

Each inode structure can hold up to twelve direct pointers, corresponding to the addresses of the file system blocks where the first twelve blocks of the file are located. If the file is too large to fit into twelve blocks (usually 12 KB or 48 KB), pointers are maintained to single, double and triple indirect pointer blocks. A single indirect pointer is the address of a file system block composed of all direct pointers. By extension, double and triple indirect pointers point to file system blocks containing single and double indirect pointers, respectively. Table 3 provides details about the structure of an ext3 inode.

Table 3. Ext3 inode structure.

Name	Bytes	Description
File Mode	2	Permission flags and file type
User ID	2	Lower 16 bits of user ID
File Size	4	Lower 32 bits of size in bytes
ACMD Times	16	Most recent access, creation, mod., del. times
Group ID	2	Lower 16 bits of group ID
Link Count	2	Number of existing links to the file
Sector Count	4	Sector occupied by the file
Flags	4	Assorted flags
Unused	4	Unused bytes
Direct Pointers	48	12 direct pointers
Single Indirect Pointer	4	1 single indirect pointer
Double Indirect Pointer	4	1 double indirect pointer
Triple Indirect Pointer	4	1 triple indirect pointer
Misc. Information	8	NFS gen. number, extended attribute block
File Size	4	Upper 32 bits of size in bytes
Fragment Information	9	Address, count and size
Unused	2	Unused bytes
User ID	2	Upper 16 bits of user ID
Group ID	2	Upper 16 bits of group ID
Unused	2	Unused bytes

Ext3 blocks are clustered into block groups. Block groups are described in a block or set of blocks called the group descriptor table, which always follows the superblock or a copy of the superblock. By default, an ext3 file system invokes a feature called a "sparse superblock," in which not every block group contains a copy of the superblock and group descriptor table; however, if these are present, they are stored in the first several blocks of the group. The number of block groups in a file system depends on the size of a block and the total size of the file system. The number of blocks in a block group is always eight times the number of bytes in a block, and the inodes are distributed evenly among the block groups.

Each block group contains two bitmap blocks, one for blocks and the other for inodes. Each bit in a bitmap reflects the status of one of the blocks or inodes in the group as allocated or unallocated. Blocks and inodes corresponding to older versions of file content or of deleted files, and those that have never been allocated are represented with a `0`. Blocks and inodes that hold current file contents or metadata, and those that are reserved by the file system are represented with a `1`.

Table 4. Default journal sizes for ext3 file systems (up to 2 GB).

File System Size	Block Size	Journal Blocks	Journal Size
< 2 MB	1,024 B	0	0 MB
2 MB	1,024 B	1,024	1 MB
32 MB	1,024 B	4,096	4 MB
256 MB	1,024 B	8,192	8 MB
512 MB	1,024 B	16,384	16 MB
513 MB	4,096 B	4,096	16 MB
1 GB	4,096 B	8,192	32 MB
2 GB	4,096 B	16,384	64 MB

3.2 Ext3 Journal

The size of the journal in an ext3 file system depends on the size of the file system (Table 4). By default, only metadata is stored in the journal.

Table 5. Ext3 journal superblock structure.

Name	Bytes	Description
Header	12	Signature (0xC03B3998), block type
Block Size	4	Size of journal block in bytes
Block Count	4	Number of blocks in journal
Start Block	4	Block where journal starts
First Transaction Sequence	4	Sequence number of the first transaction
First Transaction Block	4	Journal block of first transaction
Error Number	4	Information on errors
Features	12	Features of the journal
Journal UUID	4	Universally unique identifier of the journal
File System Count	4	Number of file systems using journal
Superblock Copy	4	Location of superblock copy
Journal Blocks per Trans.	4	Max. journal blocks per transaction
FS Blocks per Trans.	4	Max. FS blocks per transaction
Unused	176	Unused bytes
FS IDs	768	IDs of file systems using the journal

The first block in the journal always contains a special journal superblock that has information specific to the journal (Table 5). Journal entries are stored in a circular queue and each entry has one commit block and at least one descriptor block (Table 6). The descriptor block provides the sequence number of the transaction and the file system blocks being stored. If the transaction involves more blocks than can be

Table 6. Ext3 journal transaction descriptor block.

Name	Bytes	Description
Header	12	Sig. (0xC03B3998), seq. num., block type
File System Block	4	File system block where content will be written
Entry Flags	4	Same UUID, last entry in descriptor block, etc.
UUID	16	Only exists if the SAME_UUID flag is not set

described in one descriptor block, another descriptor block is created to accommodate the remaining blocks. The commit block appears only at the end of the transaction.

When a file system is in the data journaling mode, new versions of data are written to the journal before being written to disk. As with a ReiserFS journal, an ext3 journal can provide a wealth of information about the contents of old, deleted or modified files.

4. Recovering Data from Journals

A journal typically contains raw blocks that are to be written to the hard disk. These blocks may be unformatted user data or possibly blocks in the internal structure of the file system tree. Whenever a file is modified on the drive, the blocks containing raw data, either separately or in the metadata itself, are rewritten along with the metadata. Using the block map at the beginning of the journal can help determine which blocks are associated with which files.

It is important to note that a journal contains more than just deleted files. Previous versions of files can also be found as well as the nature of their recent modifications. While MAC times convey only when a file was modified, created or accessed, a journal tracks which parts of the file were modified most recently.

The following subsections discuss data recovery from journals in the Reiser and ext3 file systems, and a Java-based data recovery tool implemented for the ext3 file system.

4.1 ReiserFS Data Recovery

ReiserFS has a standard journal size of 32 MB, enough for enormous amounts of data, including documents, images and other items. Deleting a file in ReiserFS, even with a secure deletion utility likely would not purge the journal (this, of course, depends on how the utility operates).

Transactions in ReiserFS contain entire blocks of data to be written to the hard drive, serving as a cache for the computer. Furthermore, transactions held in the cache are not deleted until it is overloaded, upon which time the oldest items in the circular queue structure are overwritten. Thus, transactions may be present for quite a while, and will likely contain copies of the most recent data.

A ReiserFS journal may also contain evidence that a file was deleted and overwritten. This evidence usually manifests itself as large blocks of data (used for overwriting) that have suspicious patterns, e.g., all 0s or 1s.

4.2 Ext3 Data Recovery

In an ext3 file system with default journal settings, only changes to metadata are logged in the journal. For example, when a file is edited, the blocks logged in the journal are the primary group descriptor table, the inode of the directory entry and the directory entry of the directory that contains the file, the inode of the file, and the inode and data bitmaps of the file's block group.

In the data journaling mode, all non-journal blocks are logged when modified. Therefore, when a file is edited, all the metadata are logged along with the new file content. While this metadata can be very useful from the point of view of data recovery, only the data blocks stored in the journal are considered for our purposes. To enable the journaling of all the data, it is necessary to mount the file system with the option `data=journal` (in Linux).

We have implemented a Java-based tool that analyzes ext3 journals for information about modified, deleted and overwritten files, as well as earlier versions of files. The data recovery tool uses a `FileInputStream` object to open a file system that is passed as a command line argument and it reads the file byte by byte. Upon opening the file, information is extracted from the superblock (number of inodes, number of blocks, block size, and address of the journal inode) and stored for future use.

Based on the information stored in the journal inode, an array of `JournalBlock` objects is used to hold information about each journal block. The journal superblock is then searched to determine if each block is a descriptor, commit or data block; information about the data blocks is filled in from the data in the descriptor blocks. After this step, with the exception of entries that have had their descriptor entries overwritten, the tool discerns the type of entry of each journal block, its file system address, the sequence of which it was a part, if it was a data entry, and the file system block of which it was a copy.

Next, the tool determines whether or not each block might contain deleted content. This is accomplished by checking to see if each block was

```
mount -o loop -t reiserfs image /mnt/image
echo -e "I am writing this to tell you of secret, evil plans.\n\n\
Dr. Villain\n" >> /mnt/image/home/secret.txt

umount /mnt/image
mount -o loop -t reiserfs image /mnt/image
dd if=/dev/zero of=/mnt/image/home/secret.txt bs=1 count=128
rm -rf /mnt/image/home/secret.txt
umount /mnt/image
```

Figure 4. Script for creating, overwriting and deleting files.

a non-empty data block that was no longer in use according to the data bitmap or if the content held in the journal version of the block differed from the current version. If a block satisfies one of these conditions, the tool uses a `FileOuputStream` to write a string of header information about the block, including its file system address, the block of which it was an older version and which condition it met, along with the contents of the block.

5. Data Recovery Experiments

Experiments were performed on the Reiser and ext3 file systems to evaluate the feasibility of searching file system journals.

5.1 ReiserFS Data Recovery

A sample file system of 384 MB was created and seeded with approximately 71 MB of data using a blank file system and adding in a base Gentoo Linux 2005.1-r1 Stage 1 x86 install file set. Next the script in Figure 4 was run to create a file, overwrite it and delete it. The test system was a Gentoo Linux box running a patched 2.6.12 (with the `gentoo-r9` patch set) kernel using ReiserFS version 3.6.

Existing forensic tools are unable to recover file data without reviewing the hex dump of the entire drive. However, by examining the hex dump of the file system journal in blocks 18–8211, our data recovery tool is able to discern that the contents of the file are present in block 1407 – even though the data was overwritten with zeros (Figure 5).

Note that the block containing the directory entry is stored in the nearby journal block 1409 (Figure 6). Our data recovery tool uses this type of information to dissect journal blocks when searching for deleted or modified files.

```
0057f7f0  7f 26 e1 43 00 00 00 00  00 00 00 00 49 20 61 6d  |.&.C........I am|
0057f800  20 77 72 69 74 69 6e 67  20 74 68 69 73 20 74 6f  | writing this to|
0057f810  20 74 65 6c 6c 20 79 6f  75 20 6f 66 20 73 65 63  | tell you of sec|
0057f820  72 65 74 2c 20 65 76 69  6c 20 70 6c 61 6e 73 2e  |ret, evil plans.|
0057f830  0a 0a 44 72 20 56 69 6c  6c 61 69 6e 0a 0a 00 00  |..Dr Villain....|
0057f840  00 00 00 00 a4 81 00 00  01 00 00 00 42 00 00 00  |............B...|
0057f850  00 00 00 00 00 00 00 00  00 00 00 00 7d 00 ed 43  |............}..C|
```

Figure 5. Hex dump of ReiserFS journal block 1407.

```
005816d0  50 00 04 00 80 a9 8a 78  be 29 00 00 c2 29 00 00  |P......x.)...)..|
005816e0  40 00 04 00 73 65 63 72  65 74 2e 74 78 74 00 00  |@...secret.txt..|
005816f0  00 00 00 00 2e 6b 65 65  70 00 00 00 2e 2e 00 00  |.....keep.......|
00581700  00 00 00 00 2e 00 00 00  00 00 00 00 ed 41 00 00  |.............A..|
```

Figure 6. Hex dump of ReiserFS journal block 1409.

We wrote a tool to display the contents of a ReiserFS partition and verify deletes. Figure 7(a) shows a file (`secret.txt`) in the `/home` directory before deletion. Figure 7(b) shows the directory after the file is deleted.

The size of the journal (32 MB) precludes the ability to look too far back in the past. Specifically, when more than 32 MB of data is written to a ReiserFS Linux file system, all previously-written data is not recoverable.

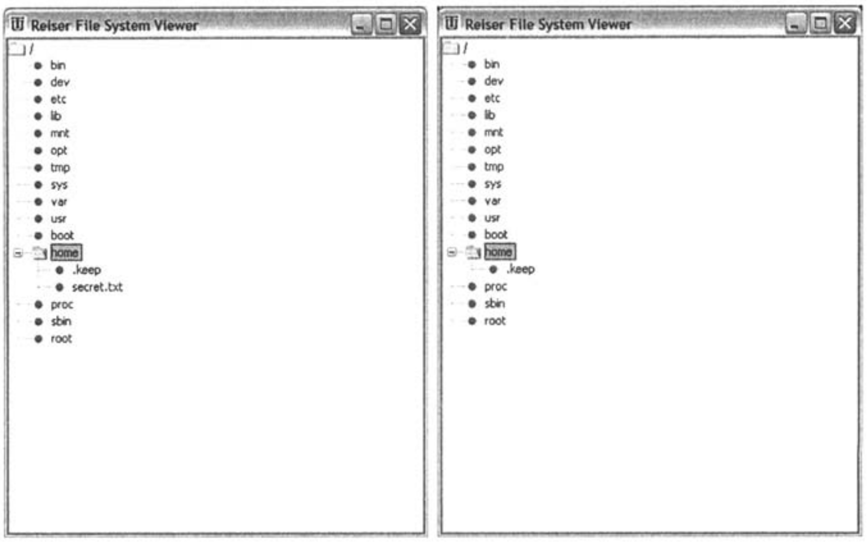
Several types of file deletions have the side effect of flushing the journal. For example, using a standard technique (e.g., issuing the command `dd if=/dev/zero`) to zero a file before deleting it will push a number of blocks containing only zeros into the journal. If a file of size greater than 32 MB is deleted in such a manner, the journal would contain practically nothing else.

Most secure deletion programs bypass an operating system's journaling constructs and directly read from or write to the hard disk. Curiously, the act of bypassing the operating system has the effect of leaving more evidence in the journal.

5.2 Ext3 Data Recovery

Data recovery in an ext3 file system was tested using a 400 MB file system created with an 8 MB journal. The test set used for ext3 was identical to the one use for ReiserFS, except that the mount command was changed to:

```
mount -o loop,data=journal -t ext3 image /mnt/image
```



(a) Before deletion (b) After deletion

Figure 7. Contents of /home directory.

The data recovery tool searched through the ext3 journal looking for possibly deleted file content. First, it determined which file system block the journal block had originally been a part of by examining the descriptor blocks and whether or not that block was a data block. The tool then dumped the contents of the block into an output file if the data allocation bit of the associated file system block was 0 or if the content was different from the current content of the file system block.

The tool was able to recover data even when files were overwritten and deleted. In the experiments, note that that the old file content is stored in journal block 7903 and the old directory entry is stored in journal block 7901. Figure 8 shows that the overwritten data is recoverable from the journal.

6. Conclusions

Because a file system journal caches data about file writes, analyzing the journal can provide valuable information about earlier versions of files. Under the right circumstances, a journal reveals information about deleted files and previous versions of files without having to review the hex dump of the drive. A journal may also yield evidence about files that are overwritten or removed by a secure deletion utility.

```
08c6000  49 20 61 6d 20 77 72 69  74 69 6e 67 20 74 68 69  |I am writing thi|
08c6010  73 20 74 6f 20 74 65 6c  6c 20 79 6f 75 20 6f 66  |s to tell you of|
08c6020  20 73 65 63 72 65 74 2c  20 65 76 69 6c 20 70 6c  | secret, evil pl|
08c6030  61 6e 73 2e 0a 0a 44 72  2e 20 56 69 6c 6c 61 69  |ans...Dr. Villai|
08c6040  6e 0a 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |n...............|
...
08c58d0  68 6f 6d 65 01 88 01 00  0c 00 03 02 73 79 73 00  |home........sys.|
08c58e0  01 10 00 00 20 03 0a 01  73 65 63 72 65 74 2e 74  |.... ...secret.t|
08c58f0  78 74 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |xt..............|
```

Figure 8. Hex dump of ext3 journal.

Despite the importance of a journal as an evidence container, few, if any, digital forensic tools consider journaled data during evidence recovery. Our Java-based tool demonstrates the feasibility of recovering data from ext3 journals, and the experimental results obtained for the Reiser and ext3 file systems are very promising.

Future work should focus on analyzing journals in NTFS and HFSJ, the default file systems for newer versions of Windows and Mac OS X, respectively. NTFS is similar to ReiserFS; moreover, NTFS's variable-sized journal feature removes certain limitations to data recovery imposed by ReiserFS [7]. However, neither NTFS nor HFSJ is as well documented as ReiserFS and ext3. Consequently, the tasks of developing forensically-sound journal data recovery and analysis tools for these file systems would be much more difficult.

File system journals contain valuable evidence pertaining to cases ranging from child pornography and software piracy to financial fraud and network intrusions. Digital forensic investigators should be aware of the data cached in file system journals and its use in digital investigations. Meanwhile, the digital forensics research community should focus its efforts on file system journal forensics and develop novel journal data extraction and analysis techniques that could be implemented in the next generation of computer forensic tools.

References

[1] F. Buchholz, The structure of the Reiser file system (homes.cerias .purdue.edu/~florian/reiser/reiserfs.php).

[2] R. Card, T. Ts'o and S. Tweedie, Design and implementation of the Second Extended File System, *Proceedings of the First Dutch International Symposium on Linux* (web.mit.edu/tytso/www/linux /ext2intro.html), 1994.

[3] B. Carrier, *File System Forensic Analysis*, Addison-Wesley, Crawfordsville, Indiana, 2005.

[4] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 2001.

[5] Fedora Project Board, Fedora Core (fedoraproject.org).

[6] G. Ganger and Y. Patt, Soft Updates: A Solution to the Metadata Update Problem in File Systems, Technical Report CSE-TR-254-95, Computer Science and Engineering Division, University of Michigan, Ann Arbor, Michigan, 1995.

[7] NTFS.com, Data integrity and recoverability with NTFS (www.ntfs.com/data-integrity.htm).

[8] S. Piper, M. Davis, G. Manes and S. Shenoi, Detecting misuse in reserved portions of ext2/3 file systems, in *Advances in Digital Forensics*, M. Pollitt and S. Shenoi (Eds.), Springer, New York, pp. 245–256, 2005.

[9] H. Reiser, ReiserFS v3 whitepaper (www.namesys.com/X0reiserfs .html), 2002.

[10] M. Rosenblum and J. Ousterhout, The design and implementation of a log-structured file system, *ACM Transactions on Computer Systems*, vol. 10(1), pp. 26–52, 1992.

[11] S. Tweedie, Journaling the Linux ext2fs filesystem, presented at the *Fourth Annual Linux Expo* (jamesthornton.com/hotlist/linux-filesystems/ext3-journal-design.pdf), 1998.

[12] S. Tweedie, Ext3: Journaling filesystem (olstrans.sourceforge.net /release/OLS2000-ext3/OLS 2000-ext3.html), July 20, 2000.

[13] U. Vahalia, C. Gray and D. Ting, Metadata logging in an NFS server, *Proceedings of the USENIX Technical Conference on Unix and Advanced Computing Systems*, pp. 265–276, 1995.