Chapter 20

DETECTING HIDDEN DATA IN EXT2/EXT3 FILE SYSTEMS

S. Piper, M. Davis, G. Manes and S. Shenoi

Abstract The use of digital forensic tools by law enforcement agencies has made it difficult for malicious individuals to hide potentially incriminating evidence. To combat this situation, the hacker community has developed anti-forensic tools that remove or hide electronic evidence for the specific purpose of undermining forensic investigations. This paper examines the latest techniques for hiding data in the popular Ext2 and Ext3 file systems. It also describes techniques for detecting hidden data in the reserved portions of these file systems.

Keywords: Anti-forensics, data hiding, file systems, Ext2/Ext3

1. Introduction

Digital forensics focuses on the identification, preservation, discovery and retrieval of electronic evidence [9]. The use of state-of-the-art forensic tools by law enforcement agencies has made it increasingly difficult for malicious individuals to hide potentially incriminating evidence. To combat this situation, the hacker community has experimented with anti-forensic techniques and tools [11]. Anti-forensics is defined as "the removal, or hiding, of evidence in an attempt to mitigate the effectiveness of a forensic investigation" [6]. Based on the number of anti-forensic tools available on the Internet [5], it is clear that anti-forensic techniques are being used by malicious individuals who wish to evade detection.

Recently, an individual known as *the grugq* unveiled the `Data Mule FS` software that conceals data within metadata structures of the Ext2 and Ext3 file systems [7]. `Data Mule FS` is specifically designed to hide data from forensic tools and file system checking software.

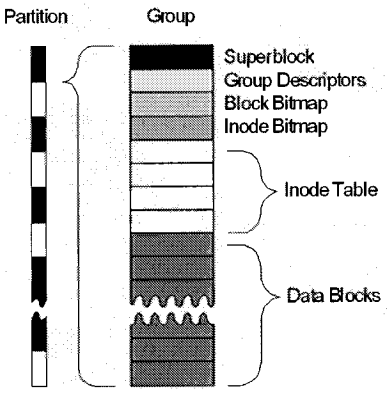

Figure 1. Ext2/Ext3 file system structure.

This paper examines the latest techniques for hiding data in the popular Ext2 and Ext3 file systems. It also describes techniques, including a utility named `rfinder`, which ensure that data hidden by anti-forensic programs such as `Data Mule FS` can be identified and retrieved by forensic investigators.

2. Ext2/Ext3 File Systems

A file system specifies how files, file metadata and other information are stored [4]. It allows the operating system to efficiently determine the locations of free space where new files can be stored. Also, it enables the operating system to quickly access and delete files.

Ext2 is the default file system [3, 15] for several Linux distributions [1]. It was created in 1993 to address limitations in earlier Linux file systems, such as Minix, which restricted file names to 14 characters and overall file system size to 64MB [3].

Ext2 has been upgraded to Ext3, which adds journaling, i.e., a means for recording hard drive accesses to minimize error-checking in the event of unexpected shutdowns [8, 13, 14]. Since Ext2 and Ext3 have the same structure, forensic techniques, anti-forensic techniques, and techniques for countering anti-forensic techniques are the same for both file systems.

Figure 1 shows the general layout of an Ext2/Ext3 file system, which comprises structures that are themselves segmented into smaller structures. The partitions are split into 8MB groups. Each group is divided into 1KB, 2KB or 4KB blocks, with the first few blocks containing metadata about the partition, group, and files within the group. The remaining blocks contain file data.

Block Bitmap (binary dump)

```
11111111 11111111 11111111 11111111 11111111 11100000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000001 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
...
11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
```

Inode Table (hex dump)

```
Inode 0 (Bad Blocks Inode)
 00001400  00 00 00 00 00 00 00 00  6c 2c 17 40 6c 2c 17 40  |........l,.@l,.@|
 00001410  6c 2c 17 40 00 00 00 00  00 00 00 00 00 00 00 00  |l,.@............|
 00001420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 00001430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 00001440  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 00001450  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 00001460  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 00001470  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
Inode 1 (Root Inode)
 00001480  ed 41 00 00 00 04 00 00  09 2e 17 40 08 2e 17 40  |.A.........@...@|
 00001490  08 2e 17 40 00 00 00 00  00 00 03 00 02 00 00 00  |...@............|
 000014a0  00 00 00 00 00 00 00 00  1c 00 00 00 00 00 00 00  |................|
 000014b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 000014c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 000014e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 000014d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
 000014f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figure 2. Block bitmap and inode blocks for a floppy disk.

The first block of the first group in a partition is called the "superblock." The superblock, which contains metadata about the partition, is essential to the operation of the file system, and its data is duplicated in superblocks in other groups for redundancy. Following the superblock are group descriptors that contain metadata about the current group and other groups in the partition (Figure 1). Specifically, group descriptors contain data about the number of free blocks in a group, and the locations of the block bitmap, inode bitmap and inode table.

Figure 2 presents a block bitmap for a floppy disk. The block bitmap indicates which blocks in the group are allocated, i.e., where file data are actually stored. Each block has a single bit in the bitmap, which is set

```
00007000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00007010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00007020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00007030  14 00 09 01 66 69 72 73  74 2e 74 78 74 00 00 00  |....first.txt...|
00007040  0d 00 00 00 14 00 0a 01  73 65 63 6f 6e 64 2e 74  |........second.t|
00007050  78 74 00 00 0e 00 00 00  ac 03 09 01 74 68 69 72  |xt..........thir|
00007060  64 2e 74 78 74 00 00 00  00 00 00 00 00 00 00 00  |d.txt...........|
00007070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Figure 3. Contents of a directory listing data block.

to one if the block is allocated and zero if it is not. Note that the bits at the end of the bitmap in Figure 2 are set to one. This is because these locations do not exist on a floppy disk; setting the corresponding bits to one (i.e., marking them as allocated) prevents the operating system from writing data to these blocks.

The inode is a smaller unit than a block that contains metadata about a particular file. Figure 2 shows a hex dump of the first two inodes of a file system. Each inode has a size of 128 bytes, and contains data about the file creation time, file size, and location of file data.

Data blocks make up the majority of the blocks in each group (typically 97%) and contain the data portions of files and directory listings [10]. Figure 3 shows the contents of a root directory. The directory listing stores the names of files/directories (., .., `lost+found`, `first.txt`, `second.txt`, `third.txt`) in the directory. Also, the directory listing stores data about the inodes corresponding to the files, and the types of the files (i.e., file or directory).

3. Data Hiding Techniques

Designed for flexibility, file system components incorporate reserved locations to store additional metadata for future upgrades to the file system. Currently, applications do not read or write to reserved locations, and data written to these locations neither overwrites any useful data nor affects system operation. Therefore, the reserved locations are ideal sites for hiding data.

Figure 4 shows the source code for a group descriptor [2]. A total of 14 bytes in the group descriptor can be used to hide data: 2 bytes for the pad (`bg_pad`) and 12 bytes for the reserved variable (`bg_reserved[3]` is an array of three 4-byte words). This may not appear to be very much space; however, such reserved locations exist throughout the file system. A large file can be hidden (e.g., by `Data Mule FS`) by dividing it into smaller portions that are inserted in each of the reserved locations.

```
struct ext2_group_desc
{
        __u32   bg_block_bitmap;        /* Blocks bitmap block */
        __u32   bg_inode_bitmap;        /* Inodes bitmap block */
        __u32   bg_inode_table;         /* Inodes table block */
        __u16   bg_free_blocks_count;   /* Free blocks count */
        __u16   bg_free_inodes_count;   /* Free inodes count */
        __u16   bg_used_dirs_count;     /* Directories count */
        __u16   bg_pad;
        __u32   bg_reserved[3];
};
```

Figure 4. Source code for the group descriptor.

Reserved locations may be discerned by reviewing the kernel source code located in `./include/linux/ext2_fs.h` [2] or the source code for `e2fsprogs`, the project that maintains and updates the Ext2 file system [12].

Hiding data within a file system does not require the modification or addition of files. On the other hand, the use of encryption or steganography to hide data is detectable by MD5 hashes and other schemes for identifying file alterations. Therefore, data concealed within file system structures is likely to go undetected.

We experimented with the ability of digital forensic tools to find data concealed within file system structures. Data was written to various reserved locations, and the images of the file systems were analyzed using three popular forensic tools: EnCase v4.15, FTK v1.42 build 03.12.05, and iLook v7.0.35. The file system checking tool, `e2fsck`, which is standard on Linux, was used to test if the file systems behaved strangely or reported errors when data was hidden in their reserved locations. The tool was run with the `-f` argument to force checking. This bypassed any shortcuts that `e2fsck` might take when file systems are flagged as healthy.

In general, neither `e2fsck` nor any of the digital forensic tools produced specific alerts when data was hidden in reserved locations. These tools assume that the reserved locations are only used for file system data and that no other data are stored within their structures. However, as we show in Section 4, these tools can be used to locate hidden data, but one must know where to look.

Data may be hidden in several reserved locations within a file system. Procedures for hiding data in four of these reserved locations are described below.

Hiding Data in the First 1KB of a Partition

Ext2 does not use the first kilobyte of each partition because it assumes that this space might be used for a boot loader. This space is not used by the file systems on floppy disks and many hard-drive partitions.

Procedure:

1. Create a file named `secret_data.txt` with size less than 1KB.
2. Insert a floppy disk into the computer.
3. Execute the command: `# mke2fs -O none /dev/fd0` to format the disk as Ext2 with no extra features.
4. Execute the command: `# dd if=secret_data.txt of=/dev/fd0` to hide the data.

Alerts: None by `e2fsck`, EnCase, FTK and iLook.

Hiding Data in Reserved Inodes

Ext2 does not use inodes 7-10, opening up 512 bytes for hiding data. On an Ext2-formatted floppy disk, hidden data may start at byte 5888. The inode blocks start at `0x1400` and the first 6 inodes take up 6*128 = `0x300` bytes, so `0x1400` + `0x300` = `0x1700` = 5888. Note that Ext3 uses inodes 7 and 8, providing 256 bytes for hidden data starting at byte 6144.

Procedure:

1. Create a file named `secret_data.txt` with size less than 512 bytes.
2. Insert a floppy disk into the computer.
3. Execute the command: `# mke2fs -O none /dev/fd0` to format the disk as Ext2 with no extra features.
4. Execute the command: `# dd if=secret_data.txt of=/dev/fd0 seek=5888` to hide the data.

Alerts: "Bad mode" error reported by `e2fsck`. None by EnCase, FTK and iLook.

Hiding Data in Redundant Superblocks

Superblocks are repeated in groups on a disk. Depending on the specific Ext2/Ext3 implementation and distribution, there is a superblock in every group or a superblock in some groups (when the "sporadic superblock" flag is set). It is possible to hide 1K to 4K of data in each redundant superblock. Note that a floppy disk has only one superblock (i.e., it has no redundant superblocks); therefore, data cannot be hidden in a floppy disk using this procedure.

Procedure:

1. Create a file named `secret_data.txt` with size between 1K to 4K (depending on the file system block size).
2. Execute the command: `# mke2fs -O none /dev/`*hd* to format a partition of hard drive *hd* (e.g., `hda1` or `hdb1`). The option `-O none` ensures that the sporadic superblock flag is not set.

3 Locate a redundant superblock by comparing blocks to a known superblock, such as the first superblock located at offset `0x400`. The redundant superblocks will also have signature values of `0x53ef` offset by `0x38` bytes from the beginning of the block.

4 Execute the command: `# dd if=secret_data.txt of=/dev/`*hd* `seek=`*loc* to hide the data. Note that *loc* is the starting byte of the redundant superblock.

Alerts: None by `e2fsck`, EnCase, FTK and iLook.

Hiding Data in Reserved Portions of Superblocks
Every superblock has at least 384 bytes reserved for updates to the file system specification. Since this group of bytes does not currently have a purpose, it can be used to hide data. Also, a superblock is supposed to fit in only 1K of space, but it uses an entire block on the file system (up to 4K). Therefore, it is often possible to hide data in the extra 3K (max) of space.

Procedure:

1 Create a file named `secret_data.txt` with size less than 384 bytes.

2 Insert a floppy disk into the computer.

3 Execute the command: `# mke2fs -O none /dev/fd0` to format the disk as Ext2 with no extra features.

4 Execute the command: `# dd if=secret_data.txt of=/dev/fd0 seek=1664` to hide the data.

Alerts: None by `e2fsck`, EnCase, FTK and iLook.

The versions of EnCase, FTK and iLook used in the experiments had the means to access and view the hidden data. However, none of the tools produced alerts that data was hidden in the reserved portions of the file systems. Forensic tools are designed to prevent investigators from having to examine data manually with a hex editor. As Figure 5 demonstrates for the EnCase tool, an investigator must use the built-in hex editor of the tool to discover the hidden data.

4. Data Detection Techniques

This section describes how data hidden in the reserved portions of the Ext2/Ext3 file systems may be detected. First, the application of digital forensic tools is discussed. Next, the use of our data detection utility (`rfinder`) is explained. Finally, the use of a file system checker is discussed.

4.1 Digital Forensic Tools

The digital forensic tools considered in this work (EnCase v4.15, FTK v1.42 build 03.12.05, and iLook v7.0.35) are not designed to discover hidden data in the reserved portions of the Ext2/Ext3 file systems. How-

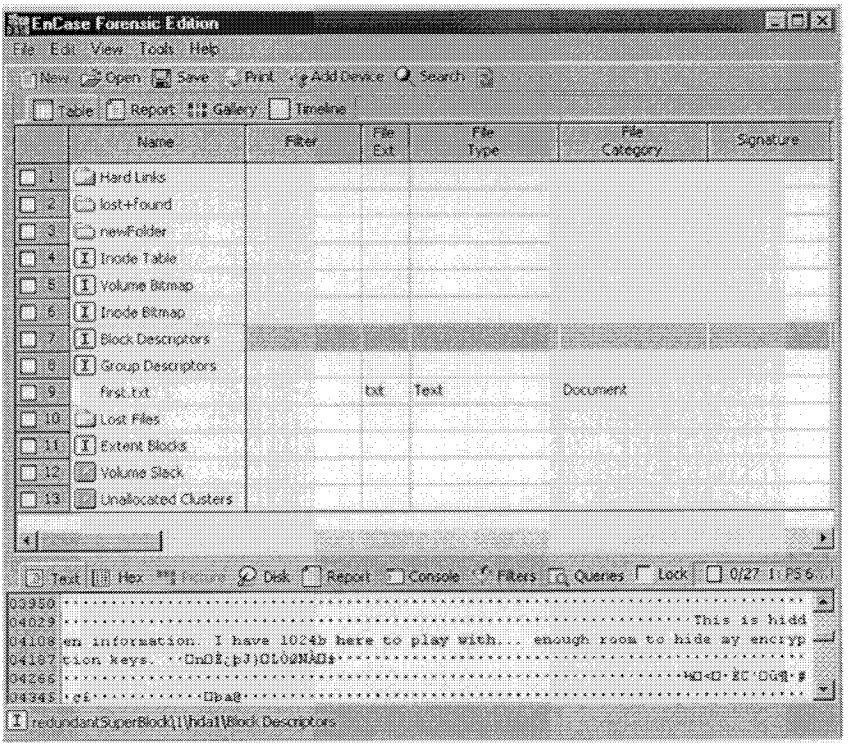

Figure 5. EnCase view of data in a redundant superblock showing hidden data.

ever, they can be used to identify hidden data if one knows where to look.

Detecting Hidden Data in the First 1KB of a Partition

Hidden data in the first 1KB of a partition can be found by looking under "Block Descriptors" for EnCase, "Boot Record" for FTK, and "Disk View" for iLook. The presence of non-zero values potentially indicates the existence of hidden data.

Detecting Hidden Data in Reserved Inodes

Hidden data in reserved inodes can be found by looking under "Inode Table" for EnCase, "Inode Table" for FTK, and "Disk View" for iLook.

Detecting Hidden Data in Redundant Superblocks

Hidden data in redundant superblocks can be found by looking under "Block Descriptors" for EnCase (see Figure 5), "Superblocks" for FTK, and "Disk View" for iLook.

Detecting Hidden Data in Reserved Portions of Superblocks
Hidden data in the reserved portions of superblocks can be found by looking under "Block Descriptors" for EnCase, "Superblocks" for FTK, and "Disk View" for iLook.

4.2 Data Detection Utility

It is possible to capitalize on the normal operation of a file system to discover hidden data. When a partition is formatted as Ext2 or Ext3, many of the reserved locations are zero wiped. This means that non-zero values that appear in the reserved locations are likely to be hidden data. We have created the `rfinder` utility that automatically searches through the reserved locations of Ext2/Ext3 file systems for non-zero values. If any non-zero values are found, `rfinder` displays a hex dump of the corresponding data.

Figure 6 shows the output obtained by running the `rfinder` utility on a 1GB partition. The partition has no hidden data in it, but if it did, a hex dump of the data would have been displayed. The hex dump of the first redundant superblock is displayed by `rfinder` because this superblock is different from the original superblock. This is common because only the first superblock is updated, for example, when the partition was last mounted. The last line of `rfinder`'s output advises the user to run the file system checker, `fsck`, on the partition to continue the search for hidden data.

4.3 File System Checkers

When the `fsck` file system checker is run on an Ext2/Ext3 partition, a utility called `e2fsck` is invoked to specifically check the Ext2/Ext3 file system. This utility can help discover whether or not other methods have been used to hide data.

The command: `e2fsck -f <image_file>` forces `e2fsck` to perform all of its checking. The image is modified slightly when `e2fsck` attempts to repair the file system. Therefore, a copy of the image must be made before executing the command. The modifications made by `e2fsck` can then be resolved using the `diff` command to compare the original and modified images. Note that `e2fsck` will flag everything that may be wrong with the file system, and not just hidden data.

Although `e2fsck` was originally created to locate faults in file systems, it is very effective for finding hidden data. For example, if certain blocks are marked as being in use even when no file uses them, `e2fsck` will identify and remedy this situation by de-allocating the blocks. This does not necessarily imply that data is hidden there. However, allocating and

```
--rfinder v0.92.2--  Author: Scott Piper

Checking /dev/hdb1...
Found an Ext2FS disk with a 4KB block size and 12 groups on the disk.

Checking the first 1K...
Nothing found (all zeroes)

Checking the superblock s_padding1 var...
Nothing found (all zeroes)

Checking the reserved portions of the first superblock (sometimes contains
        data, and will raise a false positive)...
Nothing found (all zeroes)

Checking the reserved inodes...
Nothing found (all zeroes)

Checking reserved portions of the inodes...
Nothing found (all zeroes)

Checking redundant superblock in group: 1
**This block is different the previous** (this is the first redundant super
        block though, so this is normal)
0x8000000: 20 07 03 00 60 0D 06 00 78 4D 00 00 FC F4 05 00  | ...`...xM......|
0x8000010: 15 07 03 00 00 00 00 00 02 00 00 00 02 00 00 00  |................|
0x8000020: 00 80 00 00 00 80 00 00 A0 3B 00 00 00 00 00 00  |.........;......|
0x8000030: 9D 1C F0 41 00 00 27 00 53 EF 00 00 01 00 00 00  |...A..'.S.......|
0x8000040: 9D 1C F0 41 00 4E ED 00 00 00 00 00 01 00 00 00  |...A.N..........|
0x8000050: 00 00 00 00 0B 00 00 00 80 00 01 00 00 00 00 00  |................|
0x8000060: 02 00 00 00 01 00 00 00 96 9E 7C 2F D9 EA 4D A9  |..........|/..M.|
0x8000070: BF BA 0F 8D 9A 35 D6 D1 00 00 00 00 00 00 00 00  |.....5..........|
0x8000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
0x8000090: *
0x80000E0: 00 00 00 00 00 00 00 00 00 00 00 00 AB 8B C3 27  |............'|
0x80000F0: C5 A0 44 42 85 2E 9B 9F DC 61 C6 13 02 00 00 00  |..DB.....a......|
0x8000100: 00 00 00 00 00 00 00 00 9D 1C F0 41 00 00 00 00  |...........A....|
0x8000110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  |................|
0x8000120: *
0x8000FF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00] |................|

Checking reserved group descriptor space in group: 1
Checking redundant superblock in group: 3       (same as previous)
Checking reserved group descriptor space in group: 3
Checking redundant superblock in group: 5       (same as previous)
Checking reserved group descriptor space in group: 5
Checking redundant superblock in group: 7       (same as previous)
Checking reserved group descriptor space in group: 7
Checking redundant superblock in group: 9       (same as previous)
Checking reserved group descriptor space in group: 9

Program completed successfully.
To continue looking for errors, run "fsck -f /dev/hdb1"
```

Figure 6. Output of the **rfinder** utility for a 1GB partition.

writing data to unused blocks is a technique that may be used to hide data, and `e2fsck` can help identify this situation.

5. Conclusions

Data hiding techniques have advanced from hiding secret files in folders that begin with a period (e.g., `./.hidden`) to secreting file fragments in obscure locations within file systems. Unlike other techniques, e.g., cryptography and steganography, hiding data within a file system does not involve the modification or creation of files, and is, therefore, not detectible by comparing hash values. Meanwhile, hacker tools like `Data Mule FS` are being used by malicious individuals to hide data from forensic tools and file system checking software.

The data hiding methods described in this paper are intended to expose the digital forensics community to anti-forensic techniques. Several methods for detecting hidden data are proposed, along with the `rfinder` utility, for countering anti-forensic tools such as `Data Mule FS`. While the strategies for combating anti-forensic techniques are discussed in the context of the Ext2 and Ext3 file systems, the underlying ideas are applicable to other file systems, including FAT and NTFS.

References

[1] D. Bovet and M. Cesati, *Understanding the Linux Kernel*, O'Reilly, Sebastopol, California, 2002.

[2] R. Card, Cross-referencing Linux (lxr.linux.no/source/include /linux/ext2_fs.h?v=2.6.10).

[3] 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*, 1994.

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

[5] A. Cuff, Anti-forensic tools, Computer Network Defence Ltd., Corsham, Wiltshire, United Kingdom (www.networkintrusion.co.uk/foranti.htm), 2004.

[6] The grugq, Defeating forensic analysis on Unix, *Phrack 59* (www.phrack.org/show.php?p=59&a=6), July 28, 2002.

[7] The grugq, The art of defiling, presented at the *2004 Hack in the Box Conference* (packetstormsecurity.nl/hitb04/hitb04-grugq.pdf), October 8, 2004.

[8] M. Johnson, Red Hat's new journaling file system: Ext3 (www.redhat.com/support/wpapers/redhat/ext3/index.html), 2001.

[9] W. Kruse and J. Heiser, *Computer Forensics: Incident Response Essentials*, Addison-Wesley, Boston, Massachusetts, 2002.

[10] D. Phillips, A directory index for Ext2, *Proceedings of the Fifth Annual Linux Showcase and Conference*, 2001.

[11] A. Saita, Antiforensics: The looming arms race, *Information Security Magazine*, May 2003.

[12] T. Ts'o, E2fsprogs: Ext2 file system utilities (e2fsprogs.sourceforge.net).

[13] S. Tweedie, Journaling the Linux Ext2fs filesystem, presented at the *Fourth Annual Linux Expo* (jamesthornton.com/hotlist/linux-filesystems/ext3-journal-design.pdf), 1998.

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

[15] M. Wilcox, The Second Extended File System (mail.nl.linux.org/kernel-doc/1999-03/msg00001.html), March 1, 1999.