One Big File Is Not Enough: A Critical Evaluation of the Dominant Free-Space Sanitization Technique

Simson L. Garfinkel[1] and David J. Malan[2]

[1] Center for Research on Computation and Society, Harvard University
`simsong@acm.org`

[2] Division of Engineering and Applied Sciences, Harvard University
`malan@post.harvard.edu`

Abstract. Many of today's privacy-preserving tools create a big file that fills up a hard drive or USB storage device in an effort to overwrite all of the "deleted files" that the media contain. But while this technique is widespread, it is largely unvalidated.

We evaluate the effectiveness of the "big file technique" using sector-by-sector disk imaging on file systems running under Windows, Mac OS, Linux, and FreeBSD. We find the big file is effective in overwriting file data on FAT32, NTFS, and HFS, but not on Ext2fs, Ext3fs, or Reiserfs. In one case, as much 60MB of a 488MB device was not overwritten with the technique. Also, file metadata such as filenames are rarely overwritten. We present a theoretical analysis of the file sanitization problem and evaluate the effectiveness of a commercial implementation that implements an improved strategy.

1 Introduction

It is widely known that the Unix `unlink()` and Windows `DeleteFile()` system calls do not actually overwrite the disk sectors associated with files that are "deleted." These calls merely remove the directory entries for the files from their containing directory. The file sectors are added to the list of available sectors and are overwritten only when they are allocated to other files. As a result, the contents of these "deleted" files can frequently be recovered from *free space* or *slack space* using forensic tools like EnCase [17] or The Sleuth Kit [6].[3]

While the ability to recover accidentally deleted files is useful, many users need to erase files so that recovery is not possible. For example, an individual

[3] In this paper, we use the term *free space* to describe disk sectors or clusters of disk sectors that are on the file system's "free list" and can be allocated to newly-created files. The term *slack space* refers to sectors that, while not currently allocated to files, are not on the free list. On FAT file systems, a cluster might consist of eight sectors but only the first sector might be used by a file. Because FAT allocates storage by clusters, not sectors, there is no way for the remaining seven sectors in the cluster to be allocated to a second file; these sectors are part of the slack space.

selling a laptop might need to remove confidential documents before relinquishing control of the device. Alas, the tools for deleting files provided with most computers do not satisfy this need.

One solution is to change operating systems so that files are actually overwritten when they are unlinked. A second is to provide special-purpose tools for that purpose. A third is to provide users with tools that sanitize the free space on their computers, so that already-unlinked files are actually eradicated. For this last solution, a common technique is to open a file for writing and to write one or more patterns to this file until the media is full. Figure 1 presents pseudocode for this technique.

```
procedure bigfile:
  char buf[65536]
  f = open("/volume/bigfile", "w")
  repeat until error writing file:
     write(f, buf)
  close(f)
  if (bigfile+smallfile requested) run smallfile procedure
  unlink("/volume/bigfile")
```

Fig. 1. Pseudocode for the "big file technique," which involves creating one big file that will (hopefully) overwrite all sectors currently on a volume's free list that possibly contain data from long-since "deleted" (*i.e.*, merely unlinked) files. Since some file systems limit the maximum size of a file to $2^{32} - 1$ bytes, in practice it is necessary to create multiple big files until no new files can be created, then to delete them all.

```
procedure smallfile:
  char buf[512]
  i = 0
  repeat until error opening file:
     f = open("/volume/smallfile" + i, "w")
     repeat until error writing file:
        write(f, buf)
     close(f)
     i = i + 1
```

Fig. 2. Pseudocode for the "small file technique," which involves creating numerous small files that will (hopefully) overwrite regions of the file system that are too small or fragmented to be allocated to the big file in Figure 1. This pseudocode should be run following the `close(f)` function in Figure 1 and before the `unlink()` function.

Despite the popularity of this “big file” technique, there are reasons to suspect that it leaves unscathed some sectors corresponding to deleted information. First, while the big file might expand to occupy all data sectors on the file system, in most cases it cannot overwrite file names or other metadata associated with the unlinked files. Second, the big file cannot occupy “slack space” since these sectors, by definition, are not on the free list. Third, file systems that use a log or journal to achieve high reliability might not allow the big file to overwrite the journal; overwriting the journal with user data would defeat the journal’s purpose of providing disaster recovery.

Our analysis finds that the big file technique is highly effective, but not perfect, for erasing user data on the MSDOS, FAT, and HFS file systems. The technique fails to erase many file names and other kinds of metadata. And when applied to Linux’s Ext2fs, Ext3fs, Reiserfs, and XFS file systems, the technique fails—sometimes spectacularly. We found a modified technique, which we call “Big+Small” and present in pseudocode in Figure 2, is dramatically more effective.

2 Vendor-Supplied Tools

Tools exist for Windows and Mac OS alike that claim the ability to overwrite disk sectors associated with files that have been previously deleted.

2.1 Windows’ `CIPHER.EXE`

Included with Windows XP and Windows Server 2003 (and available as a download for Windows 2000) is `CIPHER.EXE`, a command-line tool for NTFS that includes an “ability to overwrite data that you have deleted so that it cannot be recovered and accessed” [8,9]. The program’s `/w` option “[r]emoves data from available unused disk space on the entire volume.” [22]

We executed `CIPHER.EXE /W` on a 364MB ATA hard disk while tracing all of the tool’s file system activity with Filemon for Windows 7.02 [21]. During our trace, `CIPHER.EXE` appeared to:

1. Create and open a file for writing (called `\EFSTMPWP\fil2.tmp`);
2. Write 512KB at a time to the opened file in non-cached mode until the disk was nearly full;
3. Overwrite portions of `\$LogFile`, `\$BitMap`, and `\$Mft` in non-cached mode;
4. Write 512KB at a time again to the opened file in non-cached mode until one such write failed with an error indicating insufficient space;
5. Write only 512B at a time to the opened file in non-cached mode until one such write failed with an error indicating insufficient space;
6. Create and open an additional file for writing (called `\EFSTMPWP\0.E`);
7. Write 8B at a time to the new file in cached and non-cached modes until one such non-cached write failed with an error indicating insufficient space;

8. Repeat steps 6 to 7 (calling the files `\EFSTMPWP\1.E`, `\EFSTMPWP\2.E`, ...) until one such non-cached write and one such creation failed with errors indicating insufficient space;
9. Overwrite additional portions of `\$LogFile` in non-cached mode;
10. Close and delete all opened files and their containing directory;
11. Repeat steps 1 to 10 twice (calling the largest files `\EFSTMPWP\fil3.tmp` and `\EFSTMPWP\fil4.tmp`).

All the while, `CIPHER.EXE`'s output indicated only that the tool was "Writing 0x00," "Writing 0xFF," and "Writing Random Numbers."[4]

2.2 The Apple Disk Utility

Included with Mac OS 10.4 is a version of Apple Disk Utility [3] that offers the ability to "Erase Free Space" in any of three ways: "Zero Out Deleted Files," "7-Pass Erase of Deleted Files," or "35-Pass Erase of Deleted Files."[5] The tool advises that "These options erase files deleted to prevent their recovery. All files that you have not deleted are left unchanged." We were not able to trace the operation of this tool.

2.3 Third-Party Tools

Several third parties offer tools that claim the ability to wipe unallocated space thoroughly (see Section 5). We tested two such tools: SDelete 1.4 [20], which implements an algorithm that is similar to `CIPHER.EXE`'s, and Eraser 5.3 [28].

3 Experimental

We designed an experiment to evaluate the effectiveness of the big file technique for sanitizing information in free and slack space using file systems created on a "512MB"[6] Cruzer Mini USB drive manufactured by the SanDisk Corporation

[4] The technique of writing a character, its complement, and a random number is specified by the US Department of Defense Clearing and Sanitization Matrix which is present in numerous DoD publications, including DOD 5220.22-M [12].

[5] The number 35 is a reference to Gutmann's Usenix paper, "Secure Deletion of Data from Magnetic and Solid-State Memory" [18], which describes a procedure that might recover data from magnetic media after that data had been overwritten and a set of patterns which could be written to the media to make this sort of recovery more difficult. Although Gutmann has repeatedly said that there is no possible reason to use the entire 35-pass technique described in the paper, many tools nevertheless implement it.

[6] Despite the fact that the Cruzer USB drive is labeled as having "512MB" of storage, a footnote on the package revealed that the manufacturer used the letters "MB" to mean "million bytes." Most operating systems, in contrast, use the phrase "MB" to mean $1024 \times 1024 = 1,048,576$ bytes. Thus, the Cruzer Mini USB drive that we used actually had 488MB of storage.

and on a virtual disk drive of precisely the same size that was mounted as a Unix "device." We used this procedure for each experimental run:

1. Every *user addressable* sector of the device or virtual drive was cleared with the Unix `dd` command by copying `/dev/zero` to the raw device.[7]
2. The drive was formatted with the file system under study.
3. The drive was filled with one big file entirely filled with blocks of the letter "S". This file was then deleted.
4. We ran a program that we both designed and wrote called `stamp` that created a predetermined set of directories and files on the drive. Some of the directory and file names contained the letter "a" and are herein referred to as *A directories* and *A files*, while others contained the letter "b" and are herein referred to as *B directories* and *B files.*
5. The drive was unmounted and moved to an imaging workstation, where it was imaged using `aimage` [15]. The resulting image is herein referred to as the *stamped* image.
6. The drive was returned to the operating system under study, mounted, and the *B files* and *B directories* were deleted.
7. The drive was unmounted and re-imaged. The resulting image is herein referred to as the *deleted* image.
8. The drive was returned to the operating system under study, mounted, and the free-space sanitizer was run.
9. The drive was unmounted and re-imaged; the resulting image was subsequently examined for artifacts of sanitization.

It was necessary to configure Windows to treat the removable USB device as a fixed drive so that we could format the device with NTFS.[8]

To facilitate analysis, each directory and file created in the file system was given a unique name consisting of a 12-digit number and the letter "a" or "b". Files were created in a variety of file sizes from 129 to 1,798,300 bytes. A total of 80 *A files* and 80 *B files* files were placed in the root directory. In addition, a total of 10 subdirectories were created—5 *A directories* and 5 *B directories.* The *A directories* were given 80 *A files* and 80 *B files* each, while the *B directories* were given 160 *B files.* (No *A files* were placed in the *B directories* because the *B*

[7] This experiment specifically did not attempt to read previous contents of a block after it had been overwritten. For the purposes of this experiment, we assumed that once a data block was overwritten, its previous contents were gone.

[8] Although some drivers might suppress multiple writes to a disk and only write the final version of each block, this optimization would not affect our protocol as we unmounted and physically removed the Cruzer USB device prior to each imaging session. Also, while many flash storage devices employ "leveling" to ensure that individual flash cells are not overly rewritten, such leveling necessarily happens beneath the level of the block device abstraction, and not within the file system implementation. If leveling happened in the file system, then every file system would need to be specially modified in order to operate with flash devices. This is clearly not the case. To the file system, the USB device really does look like just another block-addressable device.

directories themselves were scheduled for deletion.) The contents of the files were likewise written with a recognizable pattern consisting of 512-byte records that contained the file's number and byte offset. The final record of the file included a flag indicating that it was the final record. Table 1 lists the directories and files that were written to the media as part of this "stamping" procedure.

	# entries in root	# in *A* directories	# in *B* directories	total entries in partition	# 512-byte disk sectors
A dirs	5	n/a	n/a	5	n/a
B dirs	5	n/a	n/a	5	n/a
A files	80	80 (each)	0		
		400 (total)	0	480	138,426
B files	80	80 (each)	160 (each)		
		400 (total)	800 (total)	1,280	405,865
S-filled Sectors	n/a	n/a	n/a	n/a	$\approx 450,000$[a]

Table 1. The directories and files written to each file system as part of the "stamping" procedure. The total number of 512-byte sectors is based on a calculation of file sizes made by the `stamp` program, rather than an analysis of the actual space required on the disk by the files.

[a] The actual number of "S" sectors depends on the file system overhead.

A specially written program called `report` analyzed the disk images for traces of the *B* directory names, file names, and file contents. File names were also scavenged from the disk images using `fls`, part of The Sleuth Kit [6], and the Unix `strings` command. While we were frequently able to recover all of the *B file names* and *B directory names* from our disk partitions, we were never able to recover all of the *B file contents.* This represents a minor failing of our experimental technique, but does not invalidate our primary conclusion because our technique can only err in failing to find information that is present on the disk, rather than mistaking non-information for information.

We also scanned for sectors that were filled with the letter "S". These sectors literally contained data from a previous file (the first file created) that was not allocated to any of the stamped files and could not be allocated to the sanitizing big file. That is, these sectors were part of the slack space.

3.1 Windows XP with Service Pack 2

Windows XP with Service Pack 2 supports two native file systems: FAT32 and NTFS. In each case the disk was zeroed on a Unix computer and then formatted on the Windows system using Windows' `FORMAT.EXE`.

There are two ways to delete files on Windows: they can be programmatically deleted using the `DeleteFile()` system call; or they can be deleted through the graphical user interface by dragging them to the "Recycle Bin" and then chosing to "Empty Recycle Bin," which causes each file in the Recycle Bin directory to be deleted with the system call. In our tests we deleted each file with the `DeleteFile()` system call.

We present the results for each file system in Table 2. Each column indicates the amount of metadata or data for the *B directories and files* that could be recovered using our image analysis technique. The "Data Sectors" column indicates the number of sectors from *B files* that could be recovered. (A total of 405,865 *B* sectors were written.) Since each individual block of each file was numbered, it was possible to note when a complete file could be recovered; that information is presented in the "# Complete Files" column. We classified each complete file as to whether it was "Small" (between 1 and 9 disk sectors, inclusive), "Medium" (between 10 and 99 disk sectors, inclusive), or "Large" (100 or more disk sectors). We also present the total number of complete files. Finally, the "S" sectors column indicates the number of recovered sectors that were filled with the letter "S"—this is the amount of recoverable information from the first big file that now resides in the slack space.

For each file system, the row labeled "Stamped" serves as a control for the recovery program; it shows the amount of *B* metadata and data that could be recovered by our recovery utility after the data was written to the file system but before any attempt had been made at deletion or sanitization. The row labeled "Deleted" shows the amount of metadata and program data that could be recovered after the files had been deleted with the Windows `DeleteFile()` system call. Finally, the row labeled `CIPHER.EXE /W` shows what could be recovered after Microsoft's sanitization utility was run. Similar results are reported for SDelete, Eraser, and our own big file implementation.

Image analysis shows that `CIPHER.EXE` was totally effective at overwriting free space on the FAT file system but that it left approximately 40KB of deleted file information in the case of NTFS-formatted file systems. Analysis of these sectors indicates that they are small files that were stored in the NTFS Master File Table (MFT). `CIPHER.EXE` was less successful at overwriting file names and other forms of metadata, although it was generally more successful on the FAT32 file system than on the NTFS file system.

3.2 Mac OS 10.4.4

Mac OS 10.4.4 includes native support for three file systems: Apple's Hierarchical File System (HFS), a modified version of HFS that supports journaling, and Microsoft's FAT file system (which Apple calls the "MSDOS" file system). We evaluated each; testing the FAT file system under Mac OS allowed us to see how a file system's sanitization properties are impacted by different implementations.

As with Windows, there are two ways to delete files on the Macintosh: programmatically with the `unlink()` system call and through the graphical user interface by dragging files to the Trash Can. Apple, however, has created two ways to empty the Trash Can: an "Empty Trash..." command and a "Secure Empty Trash" command (which uses Apple's user-level `srm` Secure Remove command). In this section we evaluate performance of Apple's file system with `unlink()`; we evaluate `srm` in Section 4.3.

	# *B* Metadata		# *B* Data	# Complete *B* Files				"S"
	Dirnames	Filenames	Sectors	Small	Medium	Large	Total	Sectors
FAT								
Stamped	5	1280	405,721	320	304	635	1,259	454,317
Deleted	5	480	405,721	320	304	635	1,259	454,317
`CIPHER.EXE /W`	5	480	0	0	0	0	0	1,734
NTFS								
Stamped	5	1280	405,781	293	304	635	1,232	441,514
Deleted	5	877	405,770	293	304	635	1,232	441,514
Bigfile	5	876	90	0	0	0	0	9
`CIPHER.EXE /W`	5	869	79	0	0	0	0	0
Eraser	5	292	0	0	0	0	0	0
`SDelete`	5	813	70	0	0	0	0	0

Table 2. Results of separately using the big file technique, Microsoft's `CIPHER.EXE` program, Eraser, and SDelete to sanitize the free space using Microsoft Windows with Service Pack 2 FAT and NTFS file systems. The first two columns indicate the number of deleted directory and file names recovered. The third column is the number sectors recovered from previously-deleted files. The next four columns indicate the number of complete files that could be recovered. Last is the number of unsanitizied slack sectors that recovered. Smaller numbers are better.

We hypothesized that the Erase Free Space command on Apple's MSDOS and HFS file systems would have results similar to running `CIPHER.EXE` under Windows with the FAT file system, while HFS with journaling would be similar to our results with Windows's NTFS file system.

As Table 3 shows, the big file technique was once again highly successful at erasing the free space on the partition formatted with the FAT file system. The big file was also very effective at sanitizing the HFS file system. The technique was less effective with the journaled version of HFS; presumably the few unsanitized sectors correspond to those that were in the journal. We were surprised that the `unlink()` call on the on-journaled version of HFS eradicated file names as well. We confirmed the absence of deleted file and directory names by searching for them with EnCase 5.

3.3 Linux 2.6.12

We tested an Ubuntu Linux distribution with a 2.6.12 kernel using our technique. Ubuntu comes with many file systems; we tested vfat (FAT32), Ext2fs, Ext3fs, Reiserfs 3.6, and XFS file systems. Results appear in Table 4.

Because there is no overwriting program provided with Linux, the big file technique was implemented with a specially-written program that created a single big file filled with repetitions of the letter "E". Strikingly, the big file left a

	# B Metadata		# B Data	# Complete B Files				"S"
	Dirnames	Filenames	Sectors	Small	Medium	Large	Total	Sectors
Mac OS 10.4.4 "MSDOS" (FAT)								
Stamped	5	1280	405,758	320	304	635	1,259	447,640
Deleted	5	690	405,749	319	304	635	1,258	447,640
Erase Free Space	3	480	0	0	0	0	0	0
Mac OS 10.4.4 HFS								
Stamped	5	740	405,758	320	304	635	1,259	434,296
Deleted	0	0	405,758	320	304	635	1,259	434,296
Bigfile	0	0	76	0	5	0	5	0
Erase Free Space	0	0	0	0	0	0	0	0
Mac OS 10.4.4 HFS, Journaled								
Stamped	5	740	405,760	320	304	635	1,259	417,912
Deleted	5	740	405,760	320	304	635	1,259	417,912
Bigfile	5	740	51	1	2	0	3	0
Erase Free Space	5	740	2	1	0	0	1	0

Table 3. Test results of Mac OS 10.4.4 with Apple's MSDOS, HFS, and Journaled HFS file systems. The "Bigfile" row shows the results of sanitizing the "Deleted" file system with our own program that creates a single big file, while "Erase Free Space" shows the results of sanitizing with the Mac OS 10.4.4 Disk Utility. While the big file technique does a good job overwriting the sectors associated with deleted files, Apple's Disk Utility does better.

large number of B sectors—and in many cases complete files—when applied to Ext2fs, Ext3fs, Reiserfs, and XFS file systems. With Ext2fs, more than 12% of the user data was left unsanitized by the technique, with hundreds of files being recoverable in their entirety. We saw similar performance with XFS.

We also tested the improved "big file + little file" technique with the Linux file systems. With vfat and Ext2fs the improved technique removed all of the B data sectors but still left metadata corresponding to directory names and file names. A small amount of B data was left with Ext3fs and XFS. The little file technique appeared to have no impact on Reiserfs. Not surprisingly, the effectiveness of the technique could be directly gauged by the number of little file sectors that were written.

3.4 FreeBSD 6.0

We tested FreeBSD 6.0 with the Unix File System version 2 (UFS2) and FreeBSD's support for FAT32. The big file left hundreds of complete files on the FreeBSD UFS2 file system—nearly 2MB of data on a 488MB device. The technique also left a relatively large number of complete B *files*—most of them quite small. These small files might be stored directly in the UFS inodes and thus occupy space not available to a big file.

	# *B* Metadata		# *B* Data	# Complete *B* Files				"S"
	Dirnames	Filenames	Sectors	Small	Medium	Large	Total	Sectors
Linux vfat								
Stamped	5	1280	405,758	320	304	635	1,259	447,141
Deleted	5	690	405,758	320	304	635	1,259	447,141
Bigfile	3	480	8,118	0	0	2	2	1,751
Big + Little	3	480	0	0	0	0	0	1,734
Linux Ext2fs								
Stamped	5	1280	405,758	320	304	635	1,259	394,268
Deleted	5	780	405,758	320	304	635	1,259	394,268
Bigfile	3	620	126,029	199	181	211	591	13,530
Big + Little	3	480	0	0	0	0	0	0
Linux Ext3fs								
Stamped	5	1280	405,758	320	304	635	1,259	325,214
Deleted	5	780	405,758	320	304	635	1,259	325,214
Bigfile	3	300	66,521	141	144	115	400	27,418
Big + Little	3	480	38	0	0	0	0	6
Linux Reiserfs 3.6								
Stamped	5	740	407,352	70	304	635	1,009	384,687
Deleted	5	740	407,185	70	304	635	1,009	384,688
Bigfile	5	740	1,537	17	0	0	17	20
Big + Little	5	1281	1,537	17	0	0	17	20
XFS								
Stamped	5	740	407,216	320	304	635	1,259	438,895
Deleted	5	740	407,057	320	304	635	1,259	438,895
Bigfile	5	706	119,036	100	113	229	442	183
Big + Little	5	926	1,000	1	0	0	1	59

Table 4. Results of applying our tests to the Ubuntu Linux distribution with the 2.6.12 kernel shows that the big file technique generally fails on Linux-specific file systems. Tested file systems include Linux "vfat" (FAT with long file names), Ext2fs, Ext3fs, Reiserfs, and XFS. The rows labeled "Bigfile" show the metadata and sectors left unwritten after execution of the bigfile routine, while the "Big + Little" show the amount remaining following the application of both techniques. In general, the combination of the two techniques is more effective than the big file technique alone, but it is not perfect.

4 Beyond One Big File

Although the big file technique does a good job sanitizing file content from free space, on every system we tested it fails to properly sanitize metadata. Here we evaluate the problem of free space sanitization from a theoretical prospective, discuss approaches for removing hidden information from computer systems, and evaluate the effectiveness of Apple's Secure Empty Trash file sanitizer.

	# *B* Metadata		# *B* Data	# Complete *B* Files				"S"
	Dirnames	Filenames	Sectors	Small	Medium	Large	Total	Sectors
FreeBSD "MSDOS" (FAT)								
Stamped	5	1280	405,758	320	304	635	1,259	447,776
Deleted	5	690	405,758	320	304	635	1,259	447,776
Bigfile	3	480	28	2	0	0	2	64
Big + Little	3	480	0	0	0	0	0	0
FreeBSD UFS2								
Stamped	5	1280	405,758	320	304	635	1,259	417,280
Deleted	5	960	405,758	320	304	635	1,259	417,280
Bigfile	4	640	3,863	142	103	0	245	232
Big + Little	3	800	3,239	102	86	0	188	160

Table 5. Results of testing FreeBSD 6.0 with FreeBSD's native MSDOS and UFS2 implementations shows that the big file technique largely works on the FAT file system but leaves some data behind on UFS2 file systems.

4.1 Sanitization Patterns

Garfinkel describes two design patterns or properties that can help address the problem of hidden data in computer systems:

1. **Explicit User Audit** [14, p. 325]: All user-generated information in the computer should be accessible through the computer's standard user interface, without the need to use special-purpose forensic tools.
2. **Complete Delete** [14, p. 328]: When the user attempts to delete information, the information should be overwritten so that it cannot be recovered.

These patterns apply equally well to hidden data in file systems and other data-holding structures. For example, there have been many cases in which "deleted" data has been recovered from Adobe Acrobat and Microsoft Word files [24, 25, 29]. These cases are a result of the Acrobat and Word file formats not implementing Explicit User Audit and the failure of Microsoft Word to implement Complete Delete.

As Section 3 shows, today's operating systems do not implement either of these patterns and this failing is not remedied by running existing free space and slack space sanitization tools.

4.2 Approaches for Removing Hidden Information

Let s_n be disk sector n and f be an arbitrary file. The set S_f is then the set of sectors $s_0 \ldots s_n$ that are used to hold f's data and metadata. If I is the information in file f, then the process of creating S_f could be described by:

$$S_f \leftarrow s_0 \ldots s_n \leftarrow I$$

Let S_R be the set of disk sectors that correspond to resident files and their metadata. Using this notation, the act of creating the new file f adds that files sectors to the list of resident sectors. That is,

$$S_R \leftarrow S_R \cup S_f.$$

Let S_D be the set of sectors that correspond to deleted files. In today's operating systems, deleting a file does not overwrite the information that the files contain; deleting a file simply moves that file's sectors from S_R to S_D:

$$S_R \leftarrow (S_R - S_f)$$

$$S_D \leftarrow S_D \cup S_f$$

The Explicit User Audit property can be satisfied simply by assuring that are no sectors in the file system that are both hidden and contain data. That is, we need to ensure that $S_D = \emptyset$. There are four ways to achieve this result:

1. Allow no deletion. If nothing can be deleted, then the problem of hidden dirty sectors will never arise. This approach ensures that $S_D = \emptyset$ by forbidding any modifications to S_D.
2. Have the operating system explicitly clear sectors on the target operating system before returning them to the free list. In this way is hidden data never created. (Bauer and Priyantha describe such an implementation for the Linux operating system [4].) This approach clears the sectors in S_f.
3. Create a second volume large enough to hold all resident files. Explicitly clear all sectors on the second volume,[9] then create a new file system on it.[10] Recursively copy all of the files from the root directory on the target volume to the root directory of the second volume.[11] Discard the target volume and treat the second volume as the target volume. Symbolically, this

[9] *e.g.*, `dd if=/dev/zero of=volume.iso`

[10] *e.g.*, `mdconfig -a -t vnode -f volume.iso -u 0; newfs /dev/md0`

[11] *e.g.*, `cp -pR /volume1 /volume2`

approach copies S_R to another volume and then destroys S_D. This approach is similar to a stop-and-copy garbage collection algorithm [34] and results in the only data on the new target volume being data that could be explicitly reached from the root directory of the original target volume.

4. Starting at the root directory of the target volume, recursively enumerate or otherwise mark every sector number that is used for file data or metadata. The sectors that remain will be the union of those sectors on the free list and those sectors that cannot be allocated but which do not currently hold user data. These sectors are then cleared. Symbolically, this approach clears the sectors in S_D. This approach is similar to a mark-and-sweep garbage collection algorithms [34]. Every sector that does not contain data is cleared.

These techniques have analogs when discussing data left in document files.

For example, the several cases in which confidential or classified information has leaked in Adobe Acrobat files is almost certainly a result of the way that Microsoft Word interacts with Adobe Acrobat's PDF Writer when "highlighted" words are printed. Microsoft Word allows text to be highlighted by selecting the words and then choosing the "highlight" tool from the Word formatting menu. Normally words are highlighted with the color yellow, which causes the words to stand out as if someone had colored them with a yellow "highlighter" pen. However, Word allows the color of the highlighting tool to be set by the user.

If the highlighter is set to use the color black, it can be used to redact information visually from a Microsoft Word document—that is, the information that is highlighted with black can not be seen on the computer's screen, nor will it be visible if the document is printed. An examination of the printer codes generated by Microsoft Word reveals why: Word highlights by first drawing a rectangular box in the specified highlighting color, after which it draws on top of the box. When the color black is used to highlight black text, the result is black text printed on a black background, resulting in text that cannot be discerned. However, the text is nevertheless present and can be revealed through a variety of means.

One approach for removing hidden data from a Microsoft Word document is to select and copy all of the text, then to paste the text into a new document. This technique, which was recently endorsed by the US National Security Agency [1], is similar to approach #3 above. Unfortunately, the technique does not work for included images or OLE objects, which must be handled separately. Current versions of Microsoft Office also have a "Remove Hidden Data" option in their file menu, although the mechanism of action is not documented.

4.3 Specific File Eradication Tools

An alternative to using the big file technique to sanitize disk sectors after files are deleted is to use a tool that is specifically designed to securely delete confidential information. Such tools typically use the file system `rename()` primitive to overwrite the file name and use a combination of `open()`, `write()` and `seek()`

calls to repeatedly overwrite file contents. As previously noted, these techniques may not be effective on file systems that use journals or log files.

We evaluated three such tools: SDelete's file deletion capability, Eraser's file deletion capability, and the "Secure Empty Trash" command built into Mac OS 10.4.4. We found that SDelete did a perfect job removing the B sectors containing data but that it left approximately one-sixth of the metadata associated with the B filenames. Eraser left approximately 5% of the data sectors, including 78 complete files. Mac OS "Secure Empty Trash" command also did a perfect job deleting data, but it did not delete all of the directory and file names: many could be recovered. Details appear in Table 6.

All of these commands suffer from usability problems. While free, both SDelete and Eraser are third-party programs that must be specially downloaded and run: we believe that most Windows users do not know that these commands exist. Meanwhile, the implementation of Secure Empty Trash is incomplete. Although the command appears on the Finder's File menu, it does not appear on the Trash Can's context-sensitive menu (made visible by control-clicking on the trash can). Chosing "Secure Empty Trash" locks the trash can so that it cannot be used until the operation is finished. Secure Empty Trash is very slow—performing it on the file system in Table 6 took over an hour, compared with seconds simply empty the trash. (This is a result of Apple's decision to overwrite each sector with seven passes of random data.) Finally, if the user inadvertently empties the trash, there is no way to go back and securely empty the trash.

	# B Metadata		# B Data	# Complete B Files			
	Dirnames	Filenames	Sectors	Small	Medium	Large	Total
Windows XP NTFS							
Stamped	5	1280	405,781	293	304	635	1,232
deleted with SDelete	5	246	0	0	0	0	0
deleted with Eraser	5	321	26,404	19	19	40	78
Mac OS 10.4.4 Journaled HFS							
Stamped	5	740	405,760	320	304	635	1,259
Dragged to Trash	5	740	405,760	320	304	635	1,259
Secure Empty Trash	1	45	0	0	0	0	0

Table 6. Mac OS 10.4.4 Journaled HFS with Secure Empty Trash.

5 Related Work

Although ours is the first work to vet the big file technique itself, there are several works analyzing sanitization tools.

A study by Guidance Software, authors of EnCase, found specific problems with Microsoft's `CIPHER.EXE`: "All unallocated space was filled with random values (which greatly affected file compression in the evidence file); however, the cipher tool affected only the unallocated clusters and a very small portion of the MFT; 10–15 records were overwritten in the MFT, and the majority of the records marked for deletion went untouched) [*sic*]. The utility does not affect other items of evidentiary interest on the typical NTFS partition, such as: file slack, registry files, the pagefile and file shortcuts." [30]

Geiger found defects in six counter-forensic tools [16]: Webroot Software's Window Washer 5.5 [31], NeoImagic Computing's Windows & Internet Cleaner Professional 3.60 [23], CyberScrub's CyberScrub Professional 3.5 [11], White-Canyon's SecureClean 4 [32], Robin Hood Software's Evidence Eliminator 5.0 [26], and Acronis's Acronis Privacy Expert 7.0 [2].

Burke and Craiger found similar defects [5] with Robin Hood Software's Evidence Eliminator 5.0, IDM Computer Solutions's UltraSentry 2.0 [19], CyberScrub's CyberScrub Privacy Suite 4.0, EAST Technologies' East-Tec Eraser 2005 [13], and Sami Tolvanen's Eraser 5.3 [28].

Chow *et. al.*, studied the lifetime of such sensitive data as password and encryption keys in the slack space of Unix-based computer systems using whole-system simulation. They discovered that such information, if not explicitly deleted, has a potentially indefinite lifespan [7].

One deficiency in our technique was that our stamped file systems did not contain fragmented files, because all of the files were written to the disk in a single operation. As noted by Rowe, creating realistic "fake" file systems is a non-trivial problem [27].

Finally, throughout this paper we have assumed that overwriting a sector on a hard drive with a single pass of zeros is sufficient to place the data previously in that sector beyond the possibility of recovery with conventional tools. Although Gutmann's 1996 paper discussed the possibility of recovering overwritten data using sophisticated laboratory equipment [18], the paper clearly states that the techniques only work on drives that use now-obsolete recording techniques. In a postscript added to the version of the paper that is available on the web, Gutmann states that two overwrites of random data is more than sufficient to render data irrecoverable on modern disk drives. While many researchers have claimed that a well-funded adversaries can recover overwritten data, after more than 10 years of searching we have been unable to verify or even corroborate any such claim. Crescenzo *et al*˙ also discuss techniques for overwriting secrets such as cryptographic key material. [10] In our opinion, such extraordinary measures do not seem to be warranted for the vast majority of computer users.

6 Conclusion

Clearly, there are two simple ways to erase the contents of any file system. The first is to physically destroy the storage device. The second is to erase every sector of the device using a command such as `dd`.

The technique of using one big file to sanitize the free space of an active file system has been widely implemented in many privacy-protecting and anti-forensic tools. We have found that the technique is effective at removing the contents of deleted files on FAT and NTFS file systems but that it does not reliably erase file names. The technique is less successful on many Linux file systems.

In our tests, 0.3% of the deleted data was recovered from FAT-formatted disks, no data but some file names could be recovered from Mac OS disks, and up to 17% of data could be recovered from Linux Ext2fs file systems. The big file technique can be significantly improved by creating numerous small files a sector at a time after the big file is created but before it is deleted.

The primary problem with the big file technique is that it sanitizes deleted files as a side effect of another file system operation—the operation of creating a big file. Results are inconsistent because the behavior of this side effect is not specified. "A program that has not been specified cannot be incorrect; it can only be surprising." [33]

Privacy protection should be a primary goal of modern operating systems. As such, they should give the user easy-to-use tools for deleting information. Apple's "Secure Empty Trash" is an example of such a tool, but its unnecessarily poor performance is a usability barrier to its use. A better approach would be to build this behavior directly into the `unlink()` and `DeleteFile()` system calls so that all deleted files are properly overwritten.

The test programs developed for this paper, along with the disk images that we created, can be downloaded from `http://www.simson.net/bigfile/`.

Acknowledgments

Simson L. Garfinkel is supported by a postdoctoral fellowship from the Center for Research on Computation and Society at the Division of Engineering and Applied Sciences at Harvard University. David J. Malan is funded in part by NSF Trusted Computing grant CCR-0310877.

We thank Michael D. Smith of Harvard University for his support of this work and his review of this paper. We also thank Walter Bender, Scott Bradner, Jesse Burns, Richard M. Conlan, Matthew Geiger, Peter Gutmann, Beth Rosenberg, and the anonymous reviewers for their helpful comments on this paper.

References

1. Redacting with confidence: How to safely publish sanitized reports converted from word to pdf. Technical Report I333-015R-2005, Architectures and Applications Division of the Systems and Network Attack Center (SNAC), Information Assurance Directorate, National Security Agency, 2005.
2. Acronis, Inc. `http://www.acronis.com/`.
3. Apple Computer, Inc. Apple Disk Utility, 2006.

4. Steven Bauer and Nissanka B. Priyantha. Secure data deletion for Linux file systems. In *Proc. 10th Usenix Security Symposium*, pages 153–164, San Antonio, Texas, 2001. Usenix.
5. Paul K. Burke and Philip Craiger. Digital Trace Evidence from Secure Deletion Programs. In *Proceedings of the Second Annual IFIP WG 11.9 International Conference on Digital Forensics*, Orlando, Florida, January 2006.
6. Brian Carrier. The Sleuth Kit & Autopsy: Forensics tools for Linux and other Unixes, 2005.
7. Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, and Mendel Rosenblum. Understanding data lifetime via whole system simulation. In *Proc. of the 13th Usenix Security Symposium*. Usenix, August 9–13 2004.
8. Microsoft Corporation. How To Use Cipher.exe to Overwrite Deleted Data in Windows, July 2004.
9. Microsoft Corporation. Windows 2000 Security Tool: New Cipher.exe Tool. `http://www.microsoft.com/downloads/release.asp?releaseid=30925`, March 2004.
10. Giovanni Di Crescenzo, Niels Fergurson, Russell Impagliazzo, and Markus Jakobsson. How to forget a secret. In *16th International Symposium on Theoretical Aspects of Computer Science (STACS '99)*, pages 500–509. Springer Verlag, 1999.
11. CyberScrub LLC. `http://www.cyberscrub.com/`.
12. Cleaning and sanitization matrix, January 1995. Chapter 8.
13. EAST Technologies. `http://www.east-tec.com/`.
14. Simson L. Garfinkel. *Design Principles and Patterns for Computer Systems that are Simultaneously Secure and Usable*. PhD thesis, MIT, Cambridge, MA, April 26 2005.
15. Simson L. Garfinkel, David J. Malan, Karl-Alexander Dubec, Christopher C. Stevens, and Cecile Pham. Disk imaging with the advanced forensic format, library and tools. In *Research Advances in Digital Forensics (Second Annual IFIP WG 11.9 International Conference on Digital Forensics)*. Springer, January 2006. (To appear in Fall 2006).
16. Matthew Geiger. Evaluating Commercial Counter-Forensic Tools. In *Proceedings of the 5th Annual Digital Forensic Research Workshop*, New Orleans, Louisiana, August 2005.
17. Guidance Software, Inc. EnCase Forensic.
18. Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Sixth USENIX Security Symposium Proceedings*, San Jose, California, July 22-25 1996. Usenix. Online paper has been updated since presentation in 1996.
19. IDM Computer Solutions, Inc. `http://www.ultrasentry.com/`.
20. Mark Russinovich. SDelete, 2003.
21. Mark Russinovich and Bryce Cogswell. Filemon for Windows.
22. Microsoft. Cipher.exe security tool for the encrypting file system. January 31 2006.
23. NeoImagic Computing, Inc. `http://www.neoimagic.com/`.
24. Dawn S. Onley. Pdf user slip-up gives dod lesson in protecting classified information. *Government Computer News*, 24, April 16 2005.
25. Kevin Poulsen. Justice e-censorship gaffe sparks controversy. *SecurityFocus*, October 23 2003.
26. Robin Hood Software Ltd. `http://www.evidence-eliminator.com/`.
27. Neil C. Rowe. Automatic detection of fake file systems. In *International Conference on Intelligence Analysis Methods and Tools*, May 2005.
28. Sami Tolvanen. Eraser. `http://www.tolvanen.com/eraser/`.
29. Stephen Shankland and Scott Ard. Document shows SCO prepped lawsuit against BofA. *News.Com*, March 4 2004.

30. Kimberly Stone and Richard Keightley. Can Computer Investigations Survive Windows XP? Technical report, Guidance Software, Pasadena, California, December 2001.
31. Webroot Software, Inc. `http://www.webroot.com/`.
32. WhiteCanyon, Inc. `http://www.whitecanyon.com/`.
33. W. D. Young, W. E. Boebeit, and R. Y. Kain. Proving a computer system secure. *The Scientific Honeyweller*, 6(2):18–27, July 1985. Reprinted in Computer and Network Security, M. D. Abrams and H. J. Podell, eds., IEEE Computer Security Press, 1986.
34. Benjamin Zorn. Comparing mark-and sweep and stop-and-copy garbage collection. In *LFP '90: Proceedings of the 1990 ACM conference on LISP and functional programming*, pages 87–98, New York, NY, USA, 1990. ACM Press.