CHAPTER 7

The Persistence of Deleted File Information

7.1 Introduction

Computers delete files frequently. Sometimes this happens on explicit request by a user. Often, information is deleted implicitly when an application discards some temporary file for its own internal use. Examples of such implicit file-deletion activity are text editor temporary files, files with intermediate results from program compilers, and files in Web browser caches. As you use a computer system, you unwittingly leave behind a trail of deleted information.

Computer systems have minds of their own, too, leaving their own trails of deletion as a side effect of activity that happens in the background. Examples of background deletion activity are temporary files in mail system queues or in printer queues. Such files exist for only a few seconds or minutes. If your machine provides network services to other systems, information from systems you aren't even aware of may hit your disk. Log files are another example of background file-creation and file-deletion activity.

With many computer systems, deleted file information remains intact on the disk, in unallocated data blocks and in unallocated file attribute blocks, until it is overwritten in the course of other activity. This can result in unexpected disclosure of information when a machine (or its disk) is retired and resold as secondhand equipment. For a study on how much information can be found on secondhand disk drives after the owners thought they had deleted all their files, see Garfinkel and Shelat 2003.

In this chapter, we study how deleted file information can escape destruction intact for months or even years, and how deleted file attribute information can provide insight into past system activity. We examine several systems and discover how well past activity can be preserved in unallocated disk space. At the end of the chapter, we explain why deleted file information can be more persistent than ordinary file information.

Although our results are based on UNIX file systems, we expect that they will be applicable to any modern file system that maintains a low degree of file fragmentation.

7.2 Examples of Deleted Information Persistence

In 1996, Peter Gutmann presented a paper on the problem of data destruction (Gutmann 1996), and in 2001, he delivered a follow-up paper (Gutmann 2001). Peter's concern is with the security of sensitive information such as cryptographic keys and unencrypted data. The best encryption in the world is no good when keys or unencrypted contents can be recovered.

Destroying information turns out to be difficult. Memory chips can be read even after a machine is turned off. Data on a magnetic disk can be recovered even after it has been overwritten multiple times.

Although memory chips and magnetic disks are designed to store digital information, the underlying technology is analog. With analog storage of digital information, the value of a bit is a complex combination of past stored values. Memory chips have undocumented diagnostic modes that allow access to values smaller than a bit. With modified electronic circuitry, signals from disk read heads can reveal older data as modulations on the analog signal.

Another way to examine disks is by scanning the surface. Figure 7.1 gives a spectacular example of old magnetic patterns that persist on the side of a disk track. You can find other images of semiconductors and magnetic patterns on the Veeco Web site (Veeco 2004).

However, lots of deleted information can be recovered without ever scanning the surface of magnetic disks, even when that information was deleted long ago. We examined the disk from a machine that began its life as a Windows PC, had a second life as a Solaris firewall, and finally was converted into a Linux system. After one operating system was installed over another, the deleted Solaris and Windows files were still clearly present as the contents of unallocated disk blocks. For example, we found intact copies of many deleted Solaris firewall configuration files. They could have been sitting on the machine for many more years without ever being overwritten.

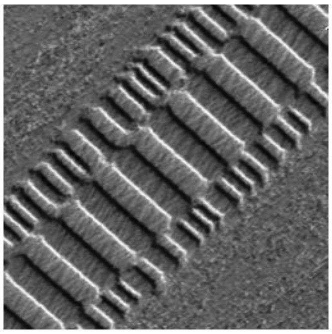

Figure 7.1 *Residuals of overwritten information on the sides of magnetic disk tracks. Reproduced with permission of Veeco.*

7.3 Measuring the Persistence of Deleted File Contents

The previous Windows PC example is unusual. We were able to estimate the age of deleted information simply because the Windows files were installed before Solaris, and because the Solaris files were installed before Linux. Although files often contain clues about when information was created, the contents of a deleted file rarely provide obvious clues about when that file was deleted.

To find out how long deleted file contents survive, we ran a 20-week experiment on a few machines on our own networks. We followed the history of each data block from day to day, from the time it was deleted to the time it was overwritten. Every night, an automated script examined each 1-Kbyte disk block and recorded a hash of the disk block's contents as well as the disk block's status: allocated, unallocated, or overhead such as inode (file attribute) or bitmap block.

Figure 7.2 shows the distribution of surviving file contents versus time of deletion for a small server file system. Despite significant fluctuation, the trend is clear. We found about 100 Mbytes of contents that were deleted less than a week ago, while about 10 Mbytes were left over from contents that were deleted 20 weeks ago. At the time of the measurement, this machine handled about 1,500 e-mail messages daily (about 10 Mbytes of data) and did limited amounts of WWW, FTP, and DNS service. Logging by the mail system amounted to about 1.5 Mbytes of data each day. The file system of 8.0 Gbytes was about 50 percent full, and most of the e-mail contents and logging were automatically deleted after a short time.

With this particular machine, half the deleted file contents were overwritten after about 35 days. Table 7.1 summarizes the results for a variety of file systems. There is some variation, but differences less than a

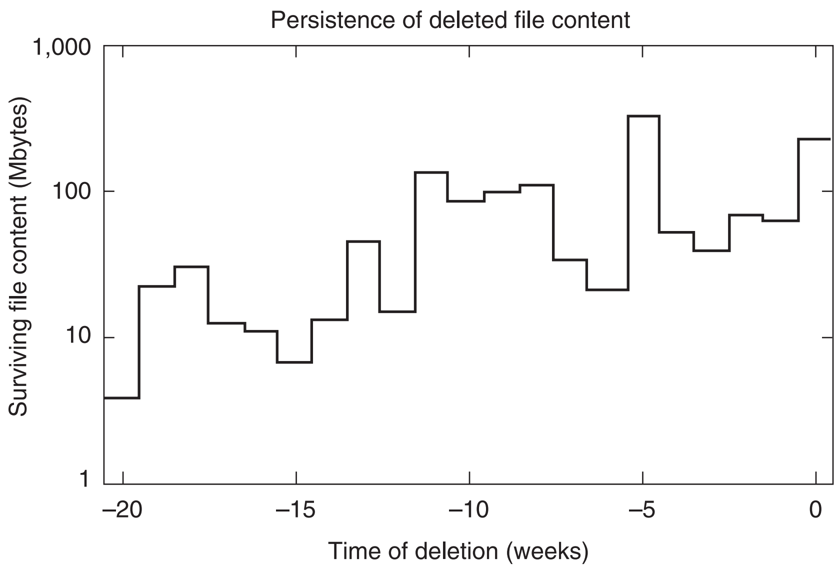

Figure 7.2 *The persistence of deleted file contents versus time of deletion for a small server file system. Time 0 corresponds to the present, and negative times represent the past. The data in the graph represent one-third of all unallocated disk blocks in the file system. The machine, spike.porcupine.org, is Wietse's FreeBSD server.*

factor of two are not significant. The lesson is that deleted data can stay around for weeks or more.

What the graph and the table do not show is how information survives. Does a deleted file slowly erode away, or does it stay mostly intact until it is finally destroyed? With file systems that suffer from fragmentation problems, we expect that a deleted file is destroyed gradually, one fragment at a time. With file systems that avoid fragmentation, we expect that a deleted file stays mostly intact until it is destroyed, in a relatively short time. We return to this topic at the end of this chapter.

Table 7.1 *The half-life of deleted file contents for three systems: spike.porcupine.org (Wietse's FreeBSD server), flying.fish.com (Dan's Linux workstation and server), and www.porcupine.org (Wietse's FreeBSD WWW and FTP server)*

Machine	File System	Half-Life
spike.porcupine.org	Entire disk	35 days
flying.fish.com	`/`	17 days
flying.fish.com	`/usr`	19 days
www.porcupine.org	Entire disk	12 days

7.4 Measuring the Persistence of Deleted File MACtimes

We recall from Chapter 2 that MACtimes, the time attributes of files, can give great insight into past activity on a machine. In this chapter, we apply the same technique to deleted file attribute information.

Furthermore, we recall from Chapter 3 that UNIX file systems store file attributes separately from file contents, and from Chapter 4 that some MACtime information survives when a file is deleted:

- The last modification time attribute (mtime) does not change (in Linux) or is set to the time of deletion (in BSD and Solaris).
- The last read access time attribute (atime) does not change.
- The last status change time attribute (ctime) is set to the time of deletion.
- Some Linux file systems have a fourth time attribute (dtime) that records when a file was deleted, but the attribute doesn't add much value; we do not discuss it further.

As we show in the next sections, deleted file attribute information can survive for months or even years, just like deleted file contents. Sometimes the reasons for survival are rather subtle, involving a combination of dumb luck and the existence of pockets of low activity in the file system. Sometimes the reasons for survival are not subtle, involving mainly the application of brute force.

7.5 The Brute-Force Persistence of Deleted File MACtimes

To find out how robust deleted file attribute information can be, we set up a disposable Linux machine and downloaded version 4 of the Linux rootkit source code, `lrk4.tgz`, from one of many malware download sites. The rootkit installs a network password sniffer program and replaces a dozen system programs with modified versions. The rootkit installation procedure uses stealth techniques to ensure that the modified program files have the same MACtimes, file sizes, and file cyclic redundancy check (CRC) values as the files being replaced. See Section 5.10 for more information about subversion with rootkit software.

We compiled the rootkit software, ran the procedure that installs the modified system utilities, and removed the rootkit source code, just as an intruder would do. Then we did just about the worst possible thing imaginable: We downloaded the Coroner's Toolkit source code distribution, unpacked the archive in the exact same directory where the "intruder" unpacked the rootkit archive, compiled our toolkit, and then ran the software in order to collect "evidence." Note: To avoid the kind of data destruction described here, the authors recommend the use of CD-ROM images with ready-to-run software. For examples, see FIRE 2004 and KNOPPIX 2004a, 2004b.

By using the Coroner's Toolkit in this manner, we knowingly destroyed large amounts of information. We overwrote data blocks that belonged to deleted rootkit files, we overwrote file attribute blocks (MACtimes!) that belonged to deleted rootkit files, and we destroyed last file access time information for compiler-related files. Kids, don't do this at home! Even after all that destruction, the Coroner's Toolkit still found the attributes of 476 deleted files and directories that existed during the rootkit incident.

In Figure 7.3, the ctime graph at the top shows the approximate times at which files were deleted. Notice the large peak on the right-hand side of the graph; this shows when the rootkit directory was removed, along with the source code and the compiler output files.

The atime graph in the middle shows when deleted files were accessed in order to compile the rootkit source code. The large atime peak on the left-hand side corresponds to rootkit files that were unpacked but not used. This is an artifact of many UNIX file systems: they set the atime of a file to the time when it is created.

The mtime graph at the bottom shows the last time that file contents were modified before they were deleted. Only 165 of the 476 deleted file residuals had mtimes in the incident time window; the data points correspond to files that were produced while compiling the rootkit source code. The remaining 311 deleted file residuals had nearly identical last file modification times in the distant past. Presumably, that was the time when the rootkit source code was packaged for distribution on some other machine.

The signal of surviving deleted file MACtimes was so strong that it should be hard to miss for anyone who knows what to look for, even days after the event. The reason for the strong signal is that rootkit software, just like other software, suffers from bloat and feature creep. Linux rootkit version 4 has a rather large total footprint of approximately 780 files and directories, including the compiler output files that are produced when the software is compiled. The Coroner's Toolkit, on the other hand, has a footprint of "only" 300 files. The number is not large enough to wipe out all the rootkit's deleted file MACtime information.

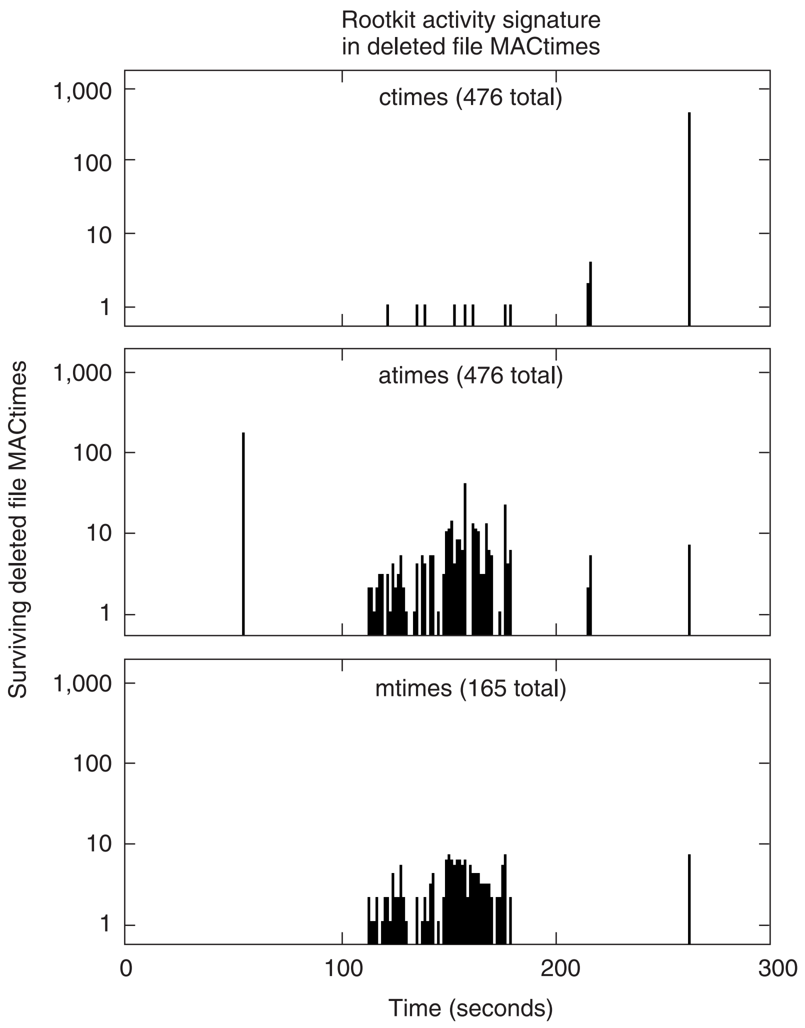

Figure 7.3 *The signature of Linux rootkit activity in deleted file MACtimes after downloading, compiling, and running the Coroner's Toolkit software. The ctime graph shows the time a file was deleted, atime shows the last read operation before a file was deleted, and mtime shows the last time the file contents were modified. See the text for a discussion of specific graph features.*

Using MACtimes for Malware Detection

MACtimes of deleted or existing files can reveal that someone may have brought specific malware into a system. Malware, like any software, is usually distributed in the form of archives that contain multiple files. The software that maintains archives carefully preserves the last modification time stamps of the original files and carefully restores those time stamps upon extraction. Even after the files are deleted, the malware's last modification time stamps can persist in the unallocated file attribute blocks.

This rootkit incident has an especially revealing signature, as shown in Figure 7.4. Of the 311 deleted file last modification times not in the incident time window, 296 were identical to within 15 seconds. Whether or not the time in the time stamps is forged does not matter. A peak with hundreds of deleted mtimes in this particular time interval should raise suspicion.

A MACtime malware signature analysis can be done relatively quickly. For example, the Coroner's Toolkit `ils` (list inodes) command can read all the 2 million file attribute blocks within an 8-Gbyte FreeBSD file system in less than half a minute, much less time than would be needed to examine gigabytes of data blocks.

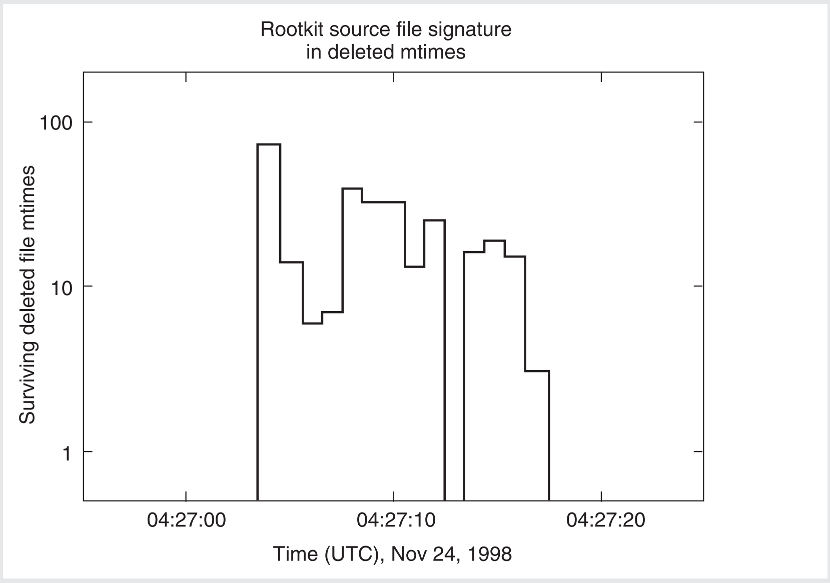

Figure 7.4 *The signature of deleted rootkit source files, revealing the apparent time and date when the source code files were packaged for distribution*

7.6 The Long-Term Persistence of Deleted File MACtimes

The brute-force persistence of deleted file MACtimes, as shown in the previous section, relies on massive file system activity in a relatively short time. This produces a strong signal that stands out well above the noise. The signal survives even when the event is followed by a significant file system activity.

The brute-force example does not tell us how long deleted file MACtime information can survive. To explore that question, we analyzed the file systems of several machines. We were surprised to find deleted file MACtime information going back an entire year or more, typically back to the time the file system was created on the disk.

Figure 7.5 shows deleted file MACtime attributes for a FreeBSD server machine that spends most of its time doing routine work: sending and receiving e-mail; providing network services such as DNS, FTP, and WWW; and maintaining log files. There is one exception to the routine. The system owner is the author of an open source mail server, and he is the “first user” of every release. “First use” involves unpacking, compiling, and removing the source code. At the time of the measurement, releases happened roughly in monthly bursts.

On the right-hand side of the figure, deleted file MACtime information decays gradually as one goes back in time. On this particular machine, 90 percent of the deleted file MACtime information is overwritten in about 60 days, as the result of routine machine activity. This corresponds with a half-life of about 20 days. This is less than the 35-day half-life found earlier for deleted file contents, but the difference is not meaningful given the accuracy of the measurements. On the left-hand side of the figure, the deleted file MACtime distributions are relatively sparse, but the patterns go back until the time that FreeBSD was installed on the machine.

The top graph, with the distribution of the ctime attribute, shows the approximate time that a file was deleted. Any deleted file ctime attributes that survive beyond the first 100 days of history are likely to be the result of nonroutine activity on the machine. For this particular machine, the most likely candidate is the compiling and installing of new mail software on the machine, and the subsequent removal of the source code.

The atime graph in the middle shows the last time that a file was accessed before it was deleted. The atime information goes back by hundreds of days, just like the graph of ctimes (file deletion times). This is not at all what one would find with ordinary file MACtimes: with ordinary files,

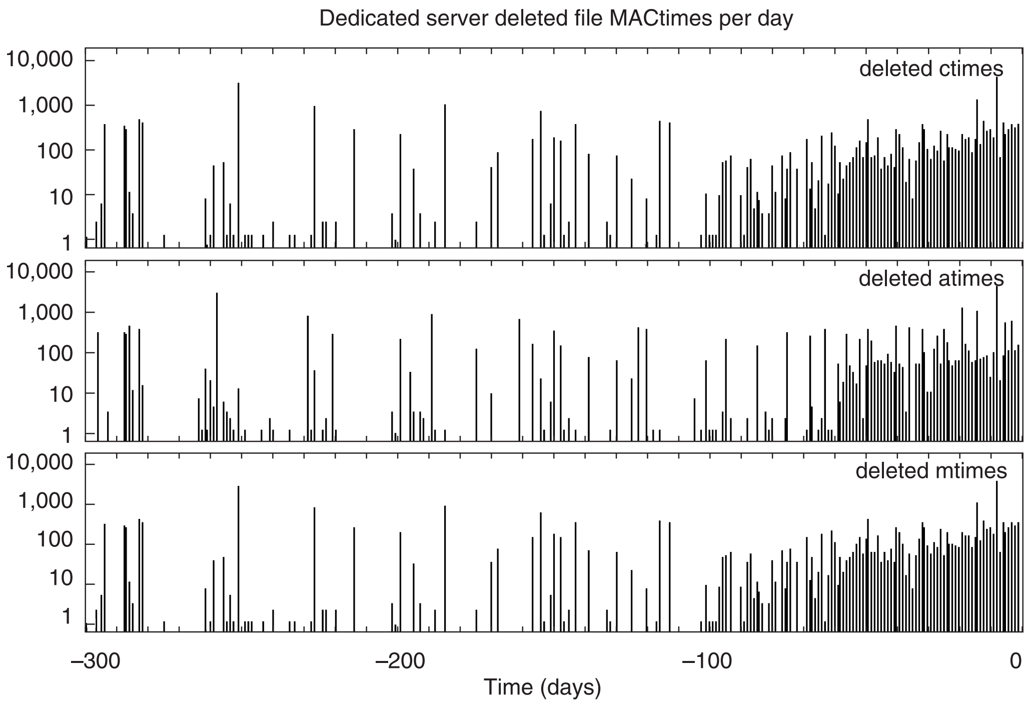

Figure 7.5 *The time distribution of deleted file MACtime attributes for a small server file system. Time 0 corresponds to the present, and negative times represent the past. The machine, spike.porcupine.org, is Wietse's FreeBSD server.*

atimes are the most volatile MACtime component. With deleted information, the rules are different: deleted file last access times are as persistent as any deleted file attribute, because they are no longer updated. We return to this phenomenon of fossilization in Section 7.10.

The bottom graph shows the distribution of the mtime attribute (file modification time). The FreeBSD file system sets the mtime to the time of deletion, and therefore its graph is identical to the ctime graph.

7.7 The Impact of User Activity on Deleted File MACtimes

Just like regular MACtimes, deleted file MACtimes are sensitive to system usage patterns. The data in the previous section are typical of a dedicated machine that spends most of its time doing routine work. The analysis of a personal workstation is more complex, because system behavior is dominated by less predictable user activity.

Figure 7.6 shows the time distribution of deleted file MACtimes for a personal workstation. This machine is the user's primary work environment for sending and receiving e-mail, surfing the Web, and developing software. In addition, it also does a limited amount of routine WWW and DNS service. The MACtime patterns for this machine are dramatically different from those of the dedicated server in Figure 7.5.

On the right-hand side, the graphs of deleted file ctimes (times of deletion) and atimes (last read access times) show decay of recent history. The decay is not nearly as smooth as in Figure 7.5. On the left-hand side, the ctime and atime graphs show residuals from significant activity in the more distant past. As with the dedicated server, the residuals go back in time until the file system was created.

The graph of the workstation's deleted file mtimes (last modification times) is unlike all the other graphs we have discussed. The workstation's distribution actually comprises two components. One component correlates with the ctime and atime graphs and corresponds to relatively

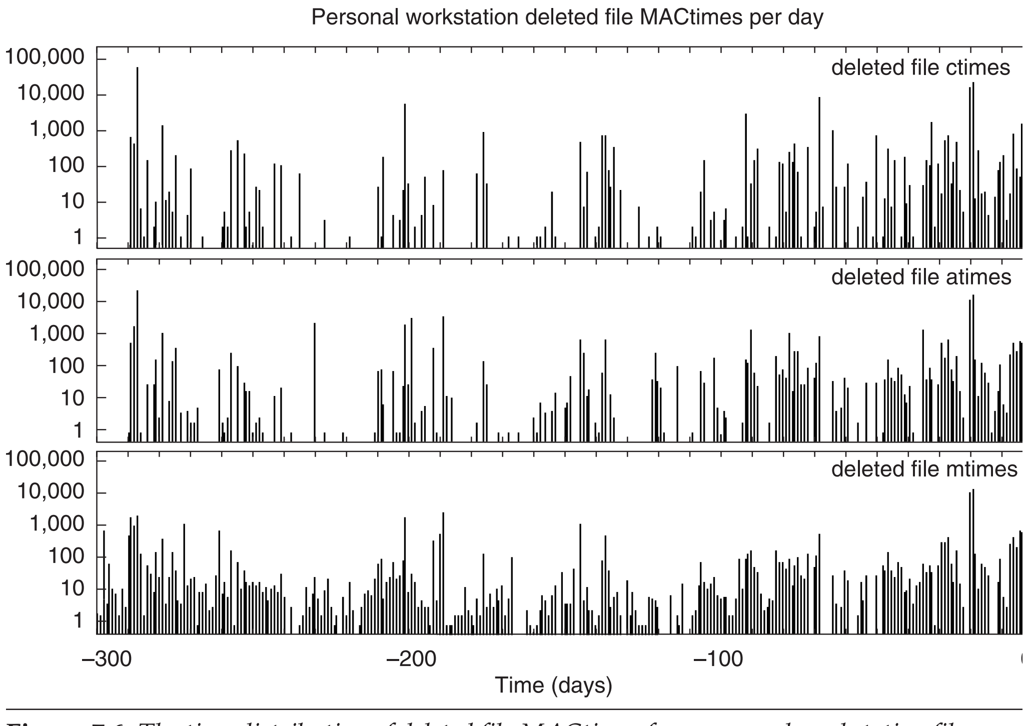

Figure 7.6 *The time distribution of deleted file MACtimes for a personal workstation file system. Time 0 corresponds to the present, and negative times represent the past. The machine flying.fish.com is Dan's Red Hat Linux workstation.*

short-lived files; the other component shows up as a more or less time-independent background of about ten deleted file residuals a day.

The existence of the time-independent component means that some files have no correlation between the time of last update and the time of deletion. This is consistent with the primary user's behavior. According to the user, files accumulate over time at a steady rate. Every few months, the user deletes a large number of files to free up some space.

7.8 The Trustworthiness of Deleted File Information

Deleted file MACtimes or contents present the investigator with great opportunities. Because deleted information is less visible than ordinary information, an opponent is less likely to be aware that the information exists, and therefore is less likely to tamper with it. For example, if a log file was modified, it is possible that portions of the unmodified file can still be recovered from unallocated file system space.

Deleted file MACtimes inherit only some of the limitations of existing file MACtimes. Prior to deletion, a file is relatively easy to access. Its MACtime information is volatile and is easily forged, as described in Chapter 2. After deletion, it is relatively easy to nonselectively overwrite deleted file MACtimes by creating a large number of small files. Changing specific deleted attributes becomes more difficult, at least on systems that can permanently revoke write access to kernel memory and disk devices (see, for example, the discussion of kernel security levels in Section 5.6).

A similar argument can be made for deleted file contents. Prior to deletion, information is relatively easy to access, and therefore relatively easy to modify. After deletion, it is relatively easy to nonselectively overwrite deleted file contents by creating a small number of large files. Changing specific deleted data blocks becomes more difficult, at least on systems that can permanently revoke write access to kernel memory and disk devices.

After deletion, forging file MACtimes or contents can be risky. The straightforward approach is to bypass the file system and write to the raw disk. There is a definite possibility of file system corruption when a mala fide opponent competes with a bona fide file system for access to the same file system block. A more reliable approach would involve a kernel module that performs the cleansing while cooperating with the file system, rather than competing against it.

Completeness is an obvious issue with deleted information. Contrary to existing file MACtimes or file contents, deleted information can be overwritten at any time, and therefore it is more likely to be incomplete. As discussed in Chapter 1, the absence of specific information must not be used as evidence that the information was never stored. With unallocated storage, this is even truer than with ordinary file information.

7.9 Why Deleted File Information Can Survive Intact

In the previous sections, we have shown that deleted information can escape destruction for months or even years. In this section, we illustrate how the design of high-performance file systems can influence the long-term survival of deleted file information.

High-performance file systems avoid disk head movements by keeping related information close together. This not only reduces the fragmentation of individual file contents, it also reduces delays while traversing directories to access a file. Although the details that follow are specific to popular UNIX systems, we expect that similar persistence effects happen with any file system that has good locality properties.

The typical UFS or Ext3fs file system is organized into equal-size zones, as shown in Figure 7.7. These file systems descend from the Berkeley Fast File System (McKusick et al. 1984) and are found on Solaris, FreeBSD, and Linux (Card et al. 1994). Typical zone sizes are 32,768 blocks; the actual block size depends on the file system type; for some systems, it also

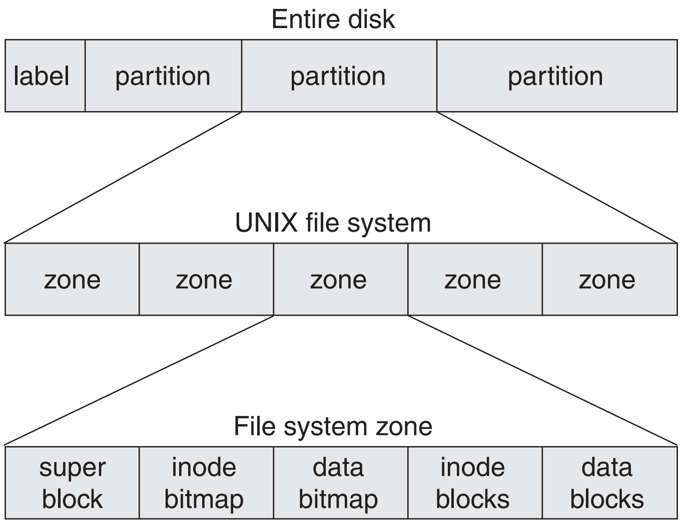

Figure 7.7 *The on-disk layout of a typical UFS or Ext3fs file system. Storage space is divided into multiple zones. Each zone contains its own allocation bitmaps, file data blocks, and file attribute (inode) blocks. Normally, information about a small file is stored entirely within one zone. The figure is not drawn to scale.*

depends on the file system size. New files are created preferably in the same file system zone as their parent directory; this improves the clustering of related information. New directories are created in zones that have few directories and lots of unused space.

By keeping related information within the same file system zone, typical UFS or Ext3fs file systems tend to cluster the files from different users or applications according to different file system zones. Because of this, the survival time of deleted information depends strongly on the amount of file write activity within its zone. As shown in Figure 7.8, write activity can be highly focused within specific file system zones.

When a file is deleted in a high-activity zone, its data blocks and file attribute information will be overwritten relatively quickly by new files. We saw an example of this in Chapter 4, when we failed to recover files that were deleted from the `/tmp` directory.

On the other hand, when a file is deleted in a low-activity zone, its data blocks and file attribute information can escape destruction as long as file system activity stays within other file system zones. As the disk fills up over time, write activity will unavoidably migrate into the quiet neighborhoods of low-activity zones, turning them into destructive, high-activity zones. Until that time, deleted file information in low-activity zones can survive intact and in copious amounts.

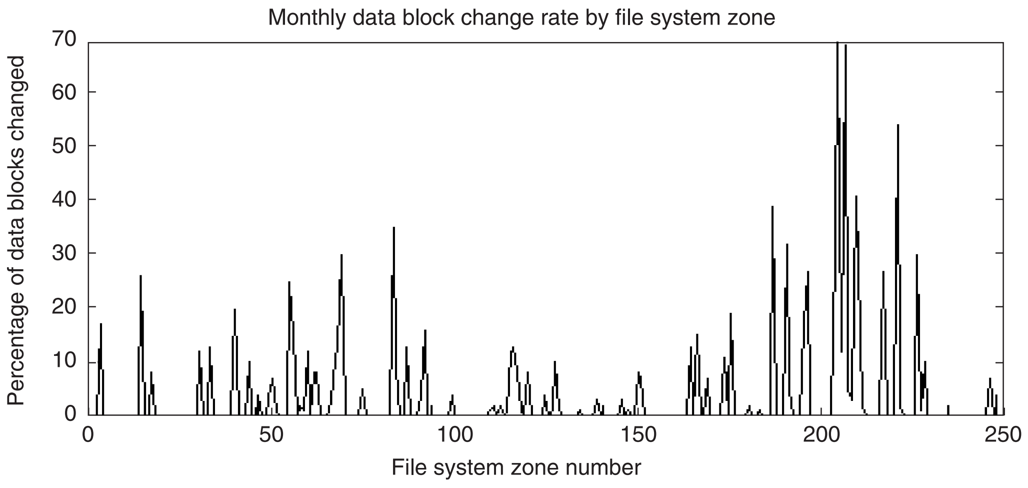

Figure 7.8 *The percentage of data blocks per file system zone that were overwritten in one month's time, for a small FreeBSD server with an 8-Gbyte file system that was filled to 50 percent capacity. The figure shows that disk write activity is focused within specific zones. Less than 4 percent of all data blocks were changed during this one-month interval.*

In Chapter 1, we observed that computer systems tend to spend most of their time running around performing routine activity. In terms of file system zones, this means that write activity tends to be focused in a limited number of zones where information is created and destroyed relatively quickly. The rest of the file system is relatively static, and any file deleted there is likely to survive for a relatively long time.

Thus, what we observed in Chapters 1 and 2 for ordinary files turns out to be true for deleted files as well: traces from routine activity erode quickly, while unusual activity stands out because its traces survive longer.

7.10 Conclusion

This chapter shows that deleted file information can survive intact for months or even years, and that deleted file attribute information can give insights about past system activity that you can't get from ordinary file attribute information.

In Chapter 1, we found that MACtime file access times for existing files can provide great insight into past system behavior. We also found that they suffer from a major drawback: MACtime information is destroyed anytime a file is accessed. Existing file MACtime information is volatile, like a footstep in sand. The next time you look, it has changed.

Deleted file MACtime information is different. When a file is deleted, its MACtime information does not change until it is overwritten. In other words, deleted file MACtime information becomes frozen in time.

The same is true for deleted file contents. Once deleted, file content does not change until it is overwritten. On file systems with good clustering properties, deleted files can remain intact for years. Deleted file information is like a fossil: a skeleton may be missing a bone here or there, but the fossil does not change until it is destroyed.

This phenomenon of deletion and persistence can happen at any abstraction level. At the abstraction level of file systems, deleted information persists as unallocated disk blocks until it is overwritten. At the abstraction level of magnetic-disk-reading heads, overwritten information persists as analog modulations on the newer information. And at the abstraction level of magnetic domains, overwritten information persists as magnetic patterns on the sides of magnetic tracks, as we saw in Figure 7.1.

At each layer in the hierarchy of abstractions that make up computer systems, information becomes frozen when it is deleted. Although deleted information becomes more and more ambiguous as we descend to lower and lower levels of abstraction, we also find that deleted information becomes ever more persistent. Volatility is an artifact of the abstractions that make computer systems useful.

All this has major consequences not only for intruders whose activity is reconstructed with post-mortem intrusion analysis, but also for the privacy of legitimate users of computer systems. For a discussion that covers much more than just computer systems, we refer the reader to Michael Caloyannides's book on privacy versus forensics (Caloyannides 2004).