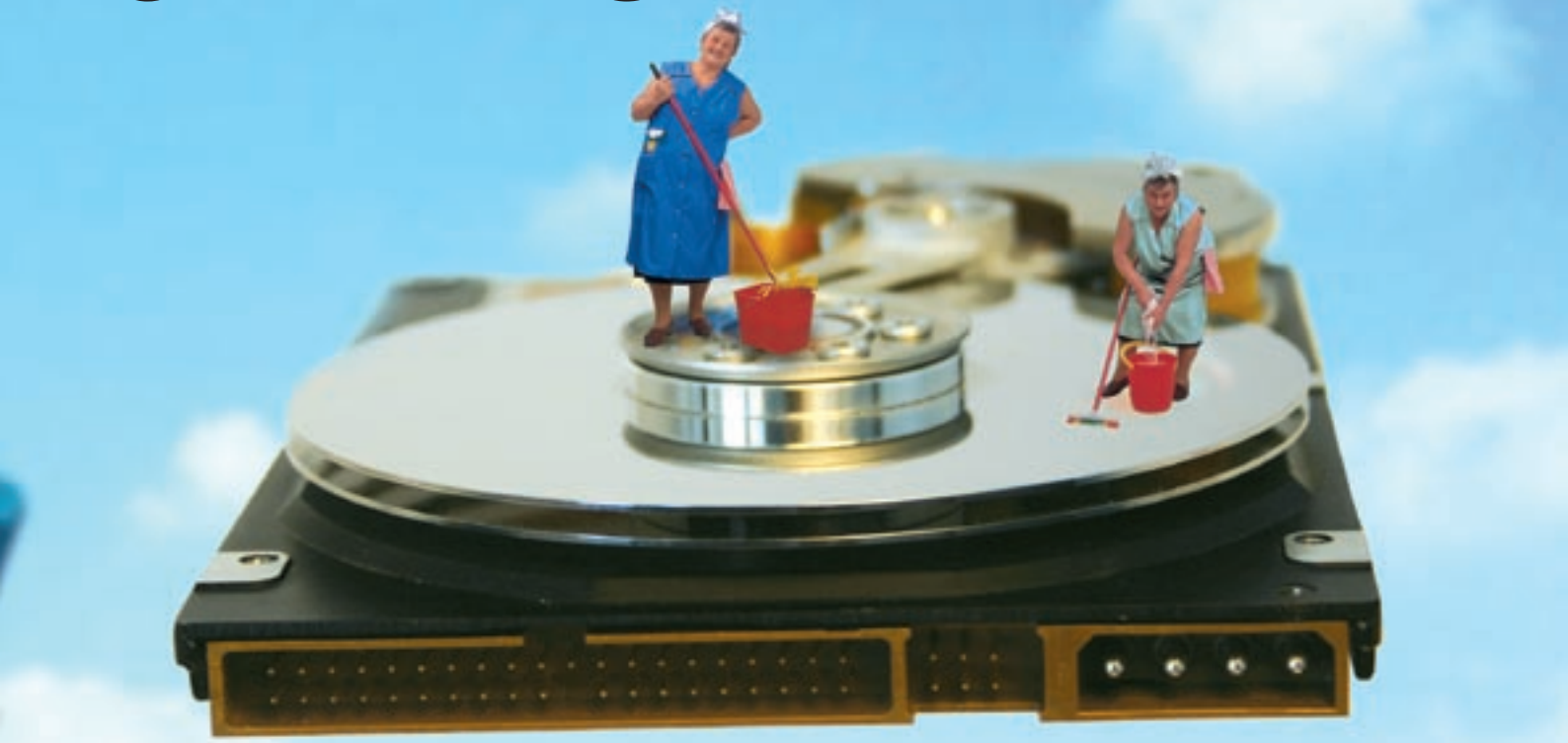
Safely deleting data

CLEANUP



Backups are a common topic, but you'll hardly hear anyone mention safe data deletion. **BY MARCUS NASAREK**

Your old hard disk is full up to the brim with data, and you are thinking about buying a new disk. To reduce the impact on your family budget, you decide to sell the old hard disk via one of the many online auction sites.

In the course of time, your whole family will have left personal traces on the disk – and of course, you would prefer to remove personal documents, letters, dismissal notices, applications, photo albums, and access credentials from the disk. Luckily, you have Linux, and a one-liner at the command line is all it takes at a pinch.

You might already know how to delete a file or a directory. GUIs all adopt similar approaches: you just click on the file or directory to select it, and then press the [Del] key to send the file off to the happy hunting grounds – or that's what you might think. Even if the application doesn't just move your files to the trash can, the content of the file is still stored somewhere on your disk in segments of various sizes. Old data doesn't actually disappear until you write new data, and start to fill up the disk again.

Typing *rm* at the command line is no help either: the *rm* command simply removes entries in the filesystem journals. Without these entries, the filesystem has no way of locating the data, but this doesn't mean that the data just disappears.

Data Deletion Background

The following command deletes the *my_data/* directory, its subdirectories, and any files they contain without prompting you to confirm.

```
rm -rf my_data/
```

Strictly speaking, what you have actually done is to remove the name tag for all these files and directories. The command simply deletes the entry in the file allocation table. Each partition needs an overview that tells it where files and directories are physically stored on the

Listing 1: Sector Search

```
01 #!/bin/bash
02 # showSector.sh
03 MAX = 1000
04 for ((i=0;i<=$MAX;i=$i+1));
05 do
06    val=`dd if=$1 skip=$i
   bs=512 count=1 2>/dev/null |
   hexdump -v -e '"%_p"' | fgrep
   $2`
07    if [ "$val" ];
08       then
09          echo -e "Sector $i:\
   n$val"
10    fi
11 done
```

disk. This approach is shared by all modern operating systems, although there are some differences in the way they store and read data.

But the important thing is that, when you "delete" a file or directory, the system simply removes the label from the table, thus freeing up some space. Physically removing the data stored in the space the label points to would mean too much computational effort.

In other words, you stand a good chance of retrieving files you deleted inadvertently if you avoid write access to the partition and start looking for the lost file straight away. You will find a description of the steps for retrieving deleted data on an Ext2 partition using simple tools at [1]. And the HOWTO at [2] gives you more details.

Even reformatting or repartitioning a drive will not really help you get rid of unwanted data. When a disk is formated, a modern operating system will just rewrite the file allocation table, and repartitioning just changes the entries in the partition table on your disk. The data may be lost somewhere in data nirvana, but it is definitely still on the disk.

Looking for Tracks

You do not need a lot of background knowledge to retrieve deleted data on a disk. The box labeled "Partitions" explains how the hard disk and the operating system manage files and directories.

If you are just looking for specific search keys or strings, such as access credentials or personal data, commands such as *dd*, *hexdump*, and *fgrep* are probably all you need. You will require *root* privileges for the disk for most of this.

A USB memory stick is a good candidate for experiments. You can format the "disk" with various operating systems, and fill it up with test data. Let's assume the device you would like to investigate is */dev/sda*; the first partition on the device would then be */dev/sda1*.

You can use the *hexdump* hex editor to browse the sectors in the partition and help you find your way around the exist-

Listing 2: So you thought you deleted it?

```
01 john@jack:~$ sudo ./showSector.sh /dev/sda1 Sesame Sector 101:
02 This is a secret. The password for the cave is "Open Sesame!".
03 But don't tell anyone!.....................................
04 ..........................................................
05 ..........................................................
06 ..........................................................
07 ..........................................................
08 ..........................................
```

Listing 3: A Safer Approach

```
01 #!/bin/bash
02 # Parameter $1 is the device to be deleted
03 BLOCKSIZE=8192
04 echo -e "Device to delete: $1\nBlock size for
   write: $BLOCKSIZE"
05 echo "[`date +"%a %T"`] Round 1"
06         d if=/dev/zero of=$1 bs=$BLOCKSIZE
07 echo "[`date +"%a %T"`] Round 2"
08         d if=/dev/urandom of=$1 bs=$BLOCKSIZE
09 echo "[`date +"%a %T"`] Round 3"
10         dd if=/dev/zero of=$1 bs=$BLOCKSIZE
11 echo "[`date +"%a %T"`] $1 deleted"
```

ing data. The following command creates a dump for the drive, and outputs any printable characters as ASCII code. You can see the content of any deleted files – apart from non-standard characters – immediately.

```
hexdump -v -e '"%07.7_ad: " ➥
60/1 "%_p" "\n"' ➥
/dev/sda1 | less
```

If this view mode is not to your liking, you might prefer to launch Midnight Commander in a console window, and to experiment with the view mode. Midnight Commander is a file manager that can also display data in hexadecimal format.

Of course, the files will not necessarily be stored in contiguous sectors on the disk; in fact, they might be spread all over the disk; experts refer to this as fragmentation. It is all a matter of how the target operating system organizes its storage space to achieve best possible read and write access times. But as a file will always occupy at least one cluster, you have a good chance of discovering small text files of less than 4096 bytes in one piece.

This technique is not much use as a systematic approach to searching for specific files. If you know that a certain string occurs in a file that you would like to retrieve (for example LaTeX files always start with */document*); a short script can help you restrict the search. Just pass the drive, the search key, and the number of sectors to search to the script in Listing 1. If your search returns results, you can then move on to investigating the sectors in question more closely.

Running the script shown in Listing 1 creates the output shown in Listing 2 when searching for the word *Sesame* on device */dev/sda1*.

You need the *dd* command, with the *skip* and *count* parameters, to investigate the location. *skip* lets you skip a certain number of blocks. The *count* option quits the program after the specified number of blocks. The script gives us a value for the *skip* parameter.

The *count* parameter returns the length of the area under investigation. Both values can be modified slightly to investigate the site of the match. A cleverly crafted combination of these parameters will allow you to copy the areas that you are interested in. For example, if sector 32 contains an interesting string, you could use the following command to investigate the 5 sectors surrounding this location:

```
dd if=/dev/sda1 skip=30 count=5➥
bs=512 2>/dev/null | hexdump ➥
-v -e '"%_p"' | less
```

This example just demonstrates the technique. Disk editors will probably be your tool of choice for analyzing or retrieving larger files from the raw data on a disk. A disk editor helps you navigate the masses of data and dump the interesting bits.

Professional data recovery services use surface analysis techniques to retrieve data from areas of the disk that have been rewritten. These surface analysis techniques leverage the fact that the write head may not hit the old track square, thus leaving residual data at the sides.

Safe Deletion

In the light of all this, you might prefer to use a standardized deletion method to delete data and rule out any possibility of recovery. The following are probably the best-known approaches:

- BSI guidelines for safeguarding classified documents on data processing systems (VSITR), Report No. 11, [BMI]

Pure Coincidence?

The special device files, */dev/random* and */dev/urandom*, use a kernel-based driver to generate pseudo-random numbers. The term "pseudo" is indicative of a well-known drawback in computing. Computers can't generate genuine random numbers, although they can generate more-or-less random values. This said, */dev/random* and */dev/urandom* generate numbers with a sufficient degree of randomness for most cryptographic applications on a PC.

When generating random sequences, the kernel-based random number generator draws on various internal values and attached devices to achieve sufficient entropy. Entropy expresses the degree of randomness of a sequence of numbers generated in a specific period of time.

Applications read a bytestream from the */dev/random* and */dev/urandom* files. In contrast to */dev/urandom*, */dev/random* only provides a byte if a sufficient degree of entropy has been achieved. If this is not the case, the device blocks the output until sufficient data has accumulated to ensure a good level of entropy for the values.

As it can take awhile for this to happen, the non-blocking I/O mode lets you remove the block, but this does not improve the output speed. The bytes provided by */dev/random* give you cryptographically stronger random numbers, and they are safe enough to use as longer ciphers and as high-quality key material.

/dev/urandom does not honor the level of entropy mandated by the kernel-based generator. Lower entropy does not interrupt the byte stream.

This fact makes the random numbers more "pseudo" than ever, but you have to remember that it isn't always necessary to aim for top marks in cryptography.

In fact, */dev/urandom* is fine for temporary keys, such as session keys in web sessions, for filling up disk space with noise, or for short-term authentication in challenge-response scenarios.

- the 5220.22-M standard by the US Department of Defense, [DOD]
- the Bruce Schneier algorithm, [BSA]
- the Peter Gutmann algorithm, [PGA]

The above mentioned algorithms overwrite the sectors multiple times with specific data patterns. The patterns comprise the bytes 0x00, 0xFF, and random values. In order to generate genuinely random data, and to make it impossible to subtract this data from the read signals, most approaches use random number generation.

Linux gives users a high level of security based on special device files such as */dev/urandom*, which creates simple random data. */dev/zero* gives you any number of null bytes (0x00). A combination of both gives you all the tools you need.

The box titled "Pure Coincidence?" gives you some background information on the quality of the two random number generators: */dev/random* and */dev/urandom*. Only */dev/urandom* is genuinely capable of producing a stream of random numbers for something as big as a hard disk.

The two files, */dev/zero* and */dev/urandom*, can be used in combination with one of the above mentioned approaches. For home use, running the script shown in Listing 3 should be fine. But take care to choose the right device if you want to be sure to remove the target data irretrievably.

The script in Listing 3 first fills the disk with zeros, then with random data, and then with zeros again. Three rounds are normally sufficient. On the downside, the program will take a few hours to delete an 80 GB hard disk.

Data protection officers will typically opt for seven rounds, as does the US DOD 5220.22-M standard. The Peter Gutmann algorithm – which is the most modern of our candidates from a technological point of view – mandates 35 rounds and requires a lot of patience on the part of the user. If you check the disk after completing the process, you should see nothing but zeros.

Deleting Individual Files

Something similar to the approach shown by the script in Listing 3 can theoretically be applied to delete a single file from a disk. But this program is not up to recursive deletion of whole directories, and you might prefer to look for an alternative or to enhance the program to meet your needs.

Applications such as Wipe give you more convenience. Most distributions include the tool, which lets you delete whole directory trees:

```
wipe -r directory
```

However, you can only rely on Wipe if you disable your hard disk's write cache. The program requests exclusive access to the disk for each round, as the project homepage explains [6].

Under normal circumstances you should be fine with a kernel that supports mandatory locking, assuming you remember to specify the *mand* option when mounting. If your system does not fulfill these requirements, or if the filesystem moves files that it overwrites to a different location, programs such as Wipe will be no use, or even worse, as they give you a false sense of security.

Wipe uses Peter Gutmann patterns to create the strings it uses for overwriting. To do so, it accesses the special */dev/urandom* and */dev/random* files, which provide the required level of entropy. To speed things up, Wipe also uses the Mersenne Twister pseudo-random number generator (PRNG).

Conclusions

It is interesting to note that the US Department of Defense stipulates physical destruction of magnetic media containing highly confidential data. Whenever you handle sensitive data, you should always be aware that there is no such thing as perfect software. On a brighter note, Linux tools should give most users more than enough security. ■

Partitions

A hard disk manages the data it contains in sectors of 512 bytes. Modern hard disks use Logical Block Addressing (LBA) to number the sectors in sequence. At the same time, the hard disk controls and monitors a number of different parameters and provides interfaces for accessing them.

The parameters include various temperature measurements, but most importantly, a map of bad sectors. The device tags sectors as bad and does not allow access to them. If you notice a rapid increase in the number of bad sectors, you can expect a total hard disk failure in the near future.

Sectors marked as bad are not available for normal access, although they may still contain data. Experts can use special tools to access these sectors, however, they are unlikely to contain top-secret data.

A hard disk must contain a partition, which in turn groups a certain number of sectors. The partition table points the way to the partitions. The partition table is located in the first sector, the so-called Master Boot Record (MBR), starting at byte 446 with a length of 64 bytes. The MBR also contains the boot loader.

The operating system manages the data for a partition in a kind of database that can have a table structure. This is the case with the File Allocation Table (FAT): DOS and Windows used a number of FAT variants known as FAT12, FAT16, and FAT32.

Linux supports a large number of file systems. Ext2, Ext3, and ReiserFS are common. The last two use a journal, something similar to a genuine database, to provide extremely effective entry management, which offers a number of advantages, especially with larger partitions. On the downside, this does make it more difficult to locate deleted data.

Filesystems group sectors to form clusters for better performance. Depending on your choice of filesystem, the clusters can be of different sizes. If you have a cluster size of four kbytes, a file will typically occupy an integral multiple of this number, or one cluster at least. A file containing a single byte of content thus occupies at least 4096 bytes of disk space.

INFO

[1] Retrieving deleted data on Linux: *http://wiki.yak.net/592*

[2] Retrieving data from an Ext2 file system: *http://www.faqs.org/docs/Linux-mini/Ext2fs-Undeletion.html*

[3] Guidelines for handling confidential data in US offices: *http://www.dss.mil/isec/nispom_0195.htm*

[4] Bruce Schneier's homepage: *http://www.schneier.com*

[5] Peter Gutmann's homepage: *http://www.cs.auckland.ac.nz/~pgut001*

[6] Wipe: *http://wipe.sourceforge.net*