A Fast Scheme for Recovery of Deleted Files with Evidential Recording for Digital Forensics

K. Kalajdzic[1,2] and A. Patel[2]

[1] Center for Computing Education - CCED, Alipasina 2/1, 71000 Sarajevo,
Bosnia and Herzegovina
[2] Department of Computer Science, Faculty of Information Science and Technology,
Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia
e-mail: kenan@unix.ba, whinchat@gmail.com

Abstract

In this paper we present a practical method for recovery of deleted files from a locally accessible data storage, such as an HDD, with an optional recording of evidential information about the recovery process. Our approach puts strong emphasis on the practical aspect of file recovery and evidence recording as well as the accessibility of the required tools. This makes it useful for a variety of tasks ranging from simple recovery of personal files to collection of evidence in digital forensic processes.

Keywords

File Recovery, Evidence Collection, Tracing, Digital Forensics, Linux

1. Introduction

Recovery of lost data has always been an interesting problem that challenged computer professionals as well as ordinary computer users. It has also played a very important role in forensic computing as a part of building computer crime prosecution cases. Whether accidentally or intentionally deleted, or vanished in a system failure, lost files can almost always be recovered. The extent and success of file recovery depends on circumstances under which files were lost as well as methods and technology available for their recovery.

Generally, all known data recovery methods can be divided into two categories:

1. hardware-based recovery methods,
2. software-based recovery methods.

Hardware-based data recovery methods rely on the fact that analogous signals are used for disk reading and writing operations and that magnetic media retains previously written data, even after new data has been subsequently written to the same disk area. Using specific modulation techniques, signals read from disk heads can be filtered in order to access this seemingly overwritten data (Veeravalli 1987). Some major drawbacks of the hardware data recovery techniques are a need for special equipment and expertise in working with disks, which makes them infeasible for a majority of computer users.

Software solutions, on the other hand, are more commonly used by individuals and small businesses and can prove very handy for a quick recovery of deleted files. There are three major categories of programs based on software data recovery methods:

a) programs that rely on the knowledge of the underlying filesystem;
b) programs which facilitate direct disk access without a specific knowledge of how files are stored on the disk;
c) programs which combine a) and b).

It is obvious that the first listed category of programs is specific to a particular filesystem from which files are to be recovered. This means that these programs have to understand how to interpret the filesystem metadata. One major advantage of understanding filesystem internals is easier gathering of all data blocks which have been lost through deletion.

In situations when metadata of deleted files has been lost or corrupted, the filesystem cannot provide enough information needed to retrieve lost files. In these cases direct disk access can be used as the last resort to data recovery.

The third and final category of programs for file recovery includes applications which rely on some combination of features provided by the first two described categories.

Most file recovery programs from all three categories record no or very little evidential information about the recovery process. However, in order to be able to use the recovered data in cybercrime investigations in court, the recovery process must be journalized to provide a formal trace of actions taken during data recovery.

2. Recovery and Recording Architecture

To make file recovery accessible to a large number of users, we decided to research possibilities of using the basic subset of known UNIX tools available in the default installation of most Linux distributions. Our primary focus are so-called live-CD distributions, like Knoppix (Knoppix Web Site 2009), which can be booted from removable media, such as CD-ROMs and DVD-ROMs. Live-CD distributions present a very convenient means for carrying out data recovery and digital forensics tasks, because they don't require access to local hard disk drives in order to boot the operating system. This guarantees that the data on these hard disks remains unaffected throughout the recovery process. It is important to mention here that recovered files should never be stored onto the same partition from which they are retrieved. Another storage device, such as an external hard disk drive attached to a local USB port or a disk partition on a remote computer accessible through the network, should be used to store them. Live-CD distributions also provide instant administrative access to the system, which is required for carrying out privileged operations that are part of the file recovery process. Despite all the mentioned advantages of live-CD distributions, our method is not limited to these types of systems and is applicable in most Linux and UNIX environments.

The recovery process relies on sequential disk access and the assumption that all blocks belonging to a single file are stored in a continuous series on the disk, as shown in Figure 1. Even though this is not always true, Wietse Venema (Venema 2000) and Dan Farmer (Farmer 2001) show that the attempts made by modern file systems to achieve data locality largely make this assumption reasonable.

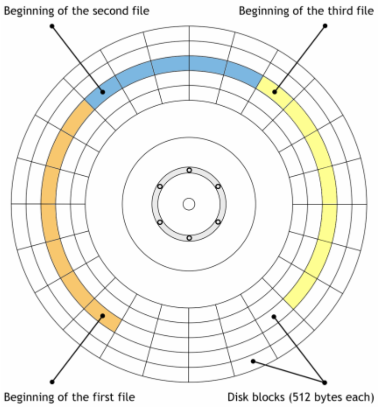
Beginning of the second file
Beginning of the third file
Beginning of the first file
Disk blocks (512 bytes each)

Figure 1: A disk partition with three files stored in series of consecutive disk blocks

Data recovery is conducted in two phases. In the first phase we read the contents of the partition which contains files we want to recover. Assuming that the beginning of every file stored on the partition is always aligned with the beginning of a disk block, we try to identify every disk block that might be the initial block of a file.

In the second phase of the file recovery process we read each disk block found during the first phase as well as its subsequent blocks, trying to piece them together and form a recoverable file.

2.1. Phase One - Finding Initial File Blocks

Most file formats define some sort of a unique signature which is stored at the very beginning of the file in order to identify the file type. This is particularly true of binary file formats. The signature consists of several bits or bytes and is often referred to as the magic number. While reading the contents of the disk, we attempt to find all blocks that contain the beginning of a file, by comparing the first few bytes of each block with the known magic number of the sought file type.

This type of comparison could be easily implemented by writing a relatively short C program. Nevertheless, in order to achieve our objective of creating a tool accessible to as many users as possible, we look for alternative ways to accomplish the same task by using already available Linux tools.

All UNIX and Linux systems have a well-known utility called `file`, which uses a large database of magic numbers and relies on some smart techniques to identify various types of files. One possibility to find the beginning of a file stored on a disk would be to use this utility to look for known magic numbers within each disk block. Even though this approach relies on common UNIX tools, and thus complies with our objective, it introduces a large overhead.

By concentrating on a small subset of interesting file types we were able to develop a much more efficient solution for finding initial file blocks. A tool called `od`, which stands for *octal dump*, plays a very important role in this process. It enables us to generate a textual dump of binary data in various formats. Since devices are accessible as files in UNIX systems, `od` can be used to easily dump the raw contents of a local disk or a partition. Converting the binary contents of the partition into an equivalent textual representation opens a possibility to use many well-known UNIX tools which operate on text files. One of these tools, used for searching textual content based on regular expressions is called `grep`.

We can use `grep` to search for magic numbers of interesting file types inside the hexadecimal dump of a partition in order to obtain a list of disk blocks which represent initial blocks of files that we are looking for. To ease this procedure, `od` is instructed to display a dump of 512 bytes per line, resulting in a direct mapping of disk blocks into lines of `od` output. Assuming that a new file always starts at the beginning of a disk block, `grep` only needs to look for the magic number within the first several bytes of each line.

Let us assume that we want to search for JPEG images stored on a disk accessible through the device file `/dev/sda`. The magic number of a JPEG image file is a two-byte sequence `0xff 0xd8`. To find all disk blocks of `/dev/sda`, which begin with this sequence, we use the following combination of `od` and `grep` commands:

```
od -Ad -tx1 -w512 /dev/sda | grep '^[^ ]* ff d8'
```

With the given combination of options, `od` is instructed to generate a hexadecimal dump of 512 bytes of data per line of output, with addresses in decimal base. Using a pipe, the output of `od` is passed as input to `grep`, which filters it by displaying only lines containing data that begins with the sought two-byte sequence, `0xff 0xd8`. The output of `grep` contains a dump of all blocks that might potentially hold the beginning of JPEG images.

```
         |------------ dump for 512 bytes ------------|

0512000 ff d8 ff e0 8d b2 b4 e7 1d 7c 8c 13 78 41 ... b2
0522240 ff d8 fe e1 d1 bc 45 69 9d 83 90 f3 ea 13 ... 9b
0644608 ff d8 b3 7a be fc c3 9e ce 45 91 4d a9 3e ... 11
...
```

Listing 1: Hexadecimal dump of disk blocks beginning with the JPEG magic number `0xff 0xd8`

The most interesting piece of information within each line of `grep` output is contained in the first field. It is the linear address of the particular block of data expressed in bytes. In Listing 1 we see the first three lines generated by combining `od` and `grep`. Notice the sequence `0xff 0xd8` at the beginning of each line.

After finding all disk blocks that begin with the JPEG magic number, we can discard the hexadecimal dump of the data, leaving only the first column with addresses.

Since every line of `od` output contains a dump for 512 bytes of data, addresses in the first column are divisible by 512. More importantly, dividing an address expressed in bytes by 512 gives the ordinal number of the corresponding disk block. In the example shown in Listing 1, the first interesting block of data is found at address **512000**, which corresponds to the disk block **1000**.

2.2. Phase Two - Recovering the Files

By determining the ordinal number of the disk block which might be the first in a series of consecutive blocks constituting a JPEG image file, we can directly access that series of blocks and try to recover the JPEG image that might be stored within. A tool called `dd` can be used to seek to a particular block on the disk and start reading data from that point. For instance, to read data from a disk accessible as `/dev/sda` in 512-byte blocks, skipping the first 1000 blocks, a user would type:

```
dd if=/dev/sda bs=512 skip=1000
```

If the block number 1000 contains the magic number of a JPEG image, the above command provides an easy way to read data stored in block 1000 and all subsequent blocks on the disk. Nevertheless, `dd` only copies data from the disk to its standard output, without having any notion of its contents. Therefore, we need another tool which is able to determine whether the data copied by `dd` is a JPEG image and, if so, where it ends.

Luckily, most Linux distributions come with a set of tools for graphic manipulations known under a common name of *Netpbm* (Netpbm Web Site 2009). These tools operate on images in the *Portable BitMap* format (thus PBM in the name). Since image files are commonly available in other formats, such as GIF, JPEG, PNG or TIFF, Netpbm contains a set of conversion utilities which enable it to work with these popular image file formats.

The Netpbm tools are designed to behave as data filters, meaning that they read original image data from the standard input and write the result to the standard output. This enables chaining of Netpbm operations by the use of pipes. These tools also behave non-destructively, by leaving the content of the original file unchanged.

The following example illustrates how we can use two Netpbm utilities to attempt a recovery of a JPEG file from a disk location we found during the first phase:

```
   dd if=/dev/sda bs=512 skip=1000 | jpegtopnm | ppmtojpeg >out.jpg
```

Using `dd`, we skip the first 1000 blocks and start reading the data from the block in which we found a JPEG magic number. A Netpbm tool called `jpegtopnm` determines whether the data sent to it by `dd` is a valid JPEG image and performs the conversion to a Netpbm format as long as it receives valid JPEG data from `dd`. As soon as `jpegtopnm` detects the end of the JPEG file or receives invalid data, it disconnects from the pipe. Consequently, `dd` stops copying data from the disk, which eventually terminates the recovery process. If a whole JPEG image was contained in the continuous set of blocks read by `dd`, `jpegtopnm` will have recovered it and converted it to the Netpbm format. A final conversion back to the JPEG format is performed by another Netbmp tool named `ppmtojpeg`.

Combining both recovery phases into the following UNIX pipeline results in a greatly simplified procedure:

```
od -Ad -tx1 -w512 /dev/sda \
| grep '^[^ ]* ff d8' \
| while read OFF REST
  do
     dd if=/dev/sda bs=512 skip=$((OFF/512)) \
     | jpegtopnm | ppmtojpeg >/remote/$OFF.jpg
  done
```

Listing 2: A single pipeline used to recover all JPEG image files from disk `/dev/sda` and store them in a directory on a remote computer

2.3. Speed Improvements

The output generated by `od` consists of the address field and a series of 512 8-bit hexadecimal values separated by spaces. Given the fact that the dump generated by `od` is sent through a pipe to `grep`, a large number of whitespace characters in the `od` output significantly degrades performance. Fortunately, it is possible to group bytes representing the data in the output, thereby reducing the number of spaces within each line. Bytes can be displayed in groups of two, four or eight, which is equivalent to four, eight and sixteen hexadecimal digits respectively.

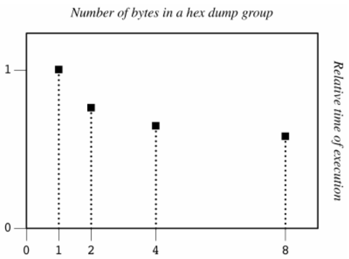

Figure 2: Speed of data transfer between od and grep for different sizes of hexadecimal dump groups

It is obvious that grouping of bytes should lead to a noticeable speed improvement in the process of passing information between `od` and `grep`, which will consequently lead to a better performance of the overall recovery process. Figure 2 shows a graph of relative speeds of data flows between `od` and `grep` for different grouping strategies.

Notice, however, that because of endianness special care must be taken when working with multi-byte groups. On little-endian architectures, the order of bytes in a multi-byte group will appear reversed when dumped by `od`, making it necessary to restructure the regular expression used for finding magic numbers within disk blocks.

2.4. Recording of the Evidence

The pipeline shown in Listing 2 implicitly records important information about the recovery process. Firstly, the name of a recovered file contains the linear disk block number in which its content was found. Secondly, the timestamp on each recovered file indicates the precise time of recovery, assuming that the clock on the remote computer (where the retrieved files are stored) is set correctly.

Besides these implicit recordings, this pipeline can be easily extended to explicitly record various traces of the recovery procedure to meet legal requirements.

One of the most important pieces of evidence is one that can guarantee the integrity of the recovered data and is able to identify investigators who worked on the recovery process (Patel and Jailani 2008). Most Linux and UNIX systems provide a tool that can be used to record such evidence. It is based on a well-known cryptographic library named *OpenSSL* (OpenSSL Web Site 2009), which supports a variety of most common cryptographic algorithms. OpenSSL can be used for symmetric and asymmetric cryptographic operations, which include encryption and decryption with ciphers, creation of RSA, DSA and Diffie-Hellman key parameters, creation of digital certificates, calculation of message digests, etc. All the functions of the OpenSSL library can be accessed through a UNIX command-line tool named `openssl`.

Assuming that each of the investigators, who are working on the recovery process, has a previously created pair of keys (e.g. RSA), `openssl` can be inserted into the pipeline from Listing 2 in order to record a digital signature of each recovered image using investigators' private keys. This would couple the investigators to the content and assure that the content is not tampered with in the period from its recovery to its presentation in court.

To implement this functionality we slightly modify the base pipeline from Listing 2. The easiest way to record tracing information is to make it part of the filename. Instead of recording just the first block number of a recovered file, the filename can hold the date and time of recovery as well. A straightforward way to achieve this is to generate the filename by using the `date` command:

```
OUTFILE=`date +"%F_%T_${OFF}.jpg"`
```

After recording the desired tracing information inside the filename, we can generate a digest of every recovered file along with the corresponding tracing information and encrypt it using a private key to produce a digital signature. One way to achieve this using OpenSSL is by using the following command sequence:

```
( echo $OUTFILE; cat $OUTFILE ) | openssl sha1 -sign private.key
```

This command couples the file contents and the corresponding tracing information and generates a single SHA-1 digest to ensure that neither is modified. It finally encrypts the digest with a private key and produces a digital signature. If the private key belongs to the investigator collecting the data, this digital signature links the investigator with the collected data and tracing information. We can add this functionality to the pipeline from Listing 2 to get a complete data recovery solution with tracing and data authenticity as shown in Listing 3.

```
od -Ad -tx1 -w512 /dev/sda \
| grep '^[^ ]* ff d8' \
| while read OFF REST
  do
     OUTFILE=`date +"%F_%T_${OFF}.jpg"`
     dd if=/dev/sda bs=512 skip=$((OFF/512)) \
     | jpegtopnm | ppmtojpeg >$OUTFILE
     ( echo $OUTFILE; cat $OUTFILE ) \
     | openssl sha1 -sign priv.key >${OUTFILE}.sig
  done
```

Listing 3: Pipeline from Listing 2 with tracing and digital signatures

The new functionality produces a signature file for each retrieved image file. To check the authenticity of the recorded data the public key of the investigator who produced the signatures is used with OpenSSL as follows:

```
F='2009-01-23_11:54:34_923254.jpg'
( echo $F; cat $F ) | openssl sha1 -verify pub.key -signature ${F}.sig
```

3. Results

We performed a number of tests of the presented recovery method on two computers with identical characteristics, equipped with 80 GB hard disk drives. The operating system used for recovery was *Knoppix v4.0.2* booted from a CD-ROM, and the retrieved files were stored over the LAN into a shared directory on another computer. Table 1 summarizes the results of our experiments.

Parameter	**Computer 1**	**Computer 2**
Total number of recovered files	8549	5522
Number of false positives	254	147
Percentage of false positives	2.97 %	2.66 %
Number of completely recovered images	7217	4855
Percentage of completely recovered images	84.4 %	87.9 %
Number of partly recovered, yet usable images	616	452
Percentage partly recovered, yet usable images	7.2 %	8.2 %
Average file size (bytes)	34680	12682
Maximum file size (bytes)	1751376	939704
Time for reading contents of disk	26 m 20 s	26 m 20 s
Time for recovery	84 m 6 s	83 m 22 s

Table 1: Experimental results of using the presented recovery method to retrieve JPEG images from 80 GB hard disk drives

4. Discussion

As the results show, the presented method leads to a very high success rate in file recovery, despite its inherent limitations and simple design. The regular expression used to search for JPEG files was limited to two bytes, resulting in a relatively high number of false positives. Fortunately, `jpegtopnm` is able to recognize that these are not valid JPEG images, so it produces no output for them. They thus appear as empty files among the recovered images, which allows us to instantly recognize and remove them. Nevertheless, by increasing the complexity and accuracy of the regular expression used to search for files of particular type, we can greatly reduce the number of false positives.

Another important result is the number of images that have been completely recovered as well as the number of partly recovered images that are still usable. By "*usable*" in the case of image files we mean that important conclusions can be drawn from the visible content, so that it can serve as evidence in case of a crime investigation. The high rate of successful recovery proves the validity of the seemingly superficial assumption that, in a large number of cases, files occupy continuous series of disk blocks (see Figure 1). Although this is commonly true of smaller files, our experiments have shown in many cases that even large files, with sizes ranging from several hundred kbytes to over one Mbyte, adhere to this rule.

Throughout this document we followed a minimalistic approach, trying to keep the number of commands required for recovery as low as possible and tie them together in a fairly simple manner. In spite of that, our choice of Linux as the recovery environment opens a great number of possibilities for easy extensions of the presented functionality.

By adding journalizing and data integrity functions in Section 2.4 we have shown that any new functionality requires only relatively small changes or additions to the pipeline shown in Listing 2, making it suitable for a bottom-up process of building complex forensic tools. This degree of extensibility makes the presented method an excellent framework for teaching file recovery and computer forensics.

Adding support for other file types requires slight modifications of the pipeline given in Listing 2. Recovery of other known image formats can be handled with Netpbm by following an equivalent procedure as illustrated on the example of JPEG images. Most Linux distributions contain tools that can be used in a similar manner for recovery of other important file types.

A big advantage of building a flexible and extensible architecture for file recovery has given us an opportunity to easily add functionality which enables recording of evidential information as a formal trace of activities performed during the process of recovery. It gives our approach an advantage over many programs which focus only on data retrieval without providing any evidence of how the recovery was performed. It should, however, be noted that the recovery time increases with every additional recording option. The total time required for recovery will depend on the complexity and number of additional commands used for evidence recording.

5. Conclusion

In this paper we demonstrated a simple versatile method for recovery of files with evidential recording, independent of file system type and based exclusively on widely available free tools found on most Linux and UNIX systems. Our approach offers most people a quick way to attempt the recovery of their lost files, but is also very important in situations where forensic analysis or data recovery must be performed on a system with restricted or no access to the outside world, which may limit the availability of more sophisticated tools. In such situations, the simple presented technique can be employed as the very first response to file recovery or forensic analysis. Through an optional recording of various types of evidence of the recovery activities, the collected files can be turned into bona fide valid evidence in computer crime investigation and prosecution cases in any jurisdiction.

By facilitating extensibility, the presented file recovery approach forms a solid base for building more complex tools that can be used in digital forensics and computer crime investigation, but can equally well be applied to other areas, such as statistics, psychology and computer science.

The straightforward nature of our technique makes it also an ideal basis for teaching. It allows students to learn systematically by building their own tools and methods around the presented concept.

6. References

Farmer, D. (2001), "Bring out your dead", http://www.ddj.com/184404444/. (Accessed February 2009)

Knoppix Web Site (2009), http://www.knoppix.org/. (Accessed February 2009)

Netpbm Web Site (2009), http://netpbm.sourceforge.net/. (Accessed February 2009)

OpenSSL Web Site (2009), http://www.openssl.org/. (Accessed February 2009)

Patel, A. and Jailani, N. (2008), "Model for Cybercrime Investigations", in *Securing Information And Communication Systems: Principles, Technologies And Applications*, Steven Furnell, Sokratis Katsikas, Javier Lopez and Ahmed Patel, Edited. Pp. 267-282. Artech House, Boston | London, 2008, 362pp. ISBN 13: 978-1-59693-228-9.

Veeravalli, V. (1987), "Detection of Digital Signals from Erased Magnetic Disks", M.S. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA, 1987.

Venema, W. (2000), "Files wanted, dead or alive", http://www.ddj.com/184404352/. (Accessed February 2009)