Data Carving Concepts

GCFA Gold Certification

Author: Antonio Merola, a.merola@securityindepth.org

Adviser: Rick Wanner

Accepted: November 10th 2008

Abstract

This paper is the technical report required to obtain the GIAC Gold for Forensics Analysis (GCFA) security certification.

From GIAC website: "The GIAC Gold option for certification is assurance that a certified individual understands and can communicate the knowledge and skills necessary in key areas of information security. GIAC Gold is distinguished from the existing exam-only 'GIAC Silver' certification by requiring candidates to complete a technical report covering an important area of security related to the certification the student is seeking."

The idea behind this paper is to help people become familiar with data carving concepts and analysis techniques.

I wrote this paper across a period of five months, the descriptions and procedures used and the conclusions obtained are as much accurate as possible, however during this periods something could have been changed a/o not still valid, please be sure to check in case you assume this possibility.

Acknowledgments

I wish to thanks Jess Garcia, instructor of the SANS track "System Forensics, Investigation & Response" his interests and knownledge are contagious. A thanks goes to the GIAC Gold Adviser Rick Wanner who reviewed this paper and, of course, to all the open-source developers for their impressive contribution to the security community.

A very special thanks to people behind the digital forensics research workshop (DFRWS). The group is dedicated to the sharing of knowledge and ideas about digital forensics research. It organized the first open workshop devoted to digital forensics in 2001, during 2006 and 2007 they released challenges about Data Carving with the goal to design and develop file carving algorithms in order to identify more files and reduce the number of false positives.

Table of Contents

1. Introduction

A proper way to start this topic, is an appropriate definition of "Data Carving" proposed by the Digital Forensic Research Workshop (DFRWS) here reported:

> **http://dfrws.org**
> "Data carving is the process of extracting a collection of data from a larger data set. Data carving techniques frequently occur during a digital investigation when the unallocated file system space is analyzed to extract files. The files are "carved" from the unallocated space using file type-specific header and footer values. File system structures are not used during the process."

Data carving is done on a disk when the unallocated file system space is analyzed to extract files because data cannot be identified due to missing of allocation info, or on network captures where files are "carved" from the dumped traffic using the same techniques. One drawback of this process on disks or images is that file-carving tools typically contain many false positives, hence tests must be done on each of the extracted files in order to check its consistency. This is not the case with network-dumped files where files are within the dump (if the dump is complete) and is just matter of doing extraction. There are many powerful automated forensic analysis tools available for use. Unfortunately, there are no standard techniques for the tools to perform common investigative tasks, such as recovering a deleted file. Having an in-depth understanding of some low level details about files, will allow analysts to evaluate forensic tools and understand the information that they present concerning data carving.

Simson Garfinkel and *Joachim Metz* proposed an interesting file carving taxonomy available online (Garfinkel, 2008). A key point in the previous definition is that the file system structure is not used during the process, this means that carving does not care which file system is used for storing files, such as:

- *FAT16, FAT32;*
- *NTFS;*
- *ext2, ext3;*
- *HFS, HFS+;*
- *ISO 9660;*

However, a knowledge of file system structure is helpful even if it is not used for data carving. A brief introduction will be provided. If you want more behind the scene details, Brian Carrier's definitive book (Carrier, 2005) is a valid choice.

2. File System

Even if data carving relies on the structure of a file, regardless of the file system where it resides, a proper introduction to File Systems (FS) may be useful. A FS is a structure for storing and organizing computer files and the data they contain to make it easy to access and find them. Some of the common file systems are: **FAT (File Allocation Table) / NTFS (New Technology File System)** on Windows Systems and **UFS/JFS** on Unix Systems. The FS software, is responsible for organizing disk sectors (typically ***512 bytes*** each) into files and directories and keeping track of which sectors belong to which file (***allocated***) and which are not being used (***unallocated***). In *Figure 1*, there is an example of a disk structure where files reside, with a tipical file allocation. FS typical have directories that associate file names with files, usually by connecting the file name to an index into a file allocation table, such as the *FAT*, or an *inode* for Unix-like file system.

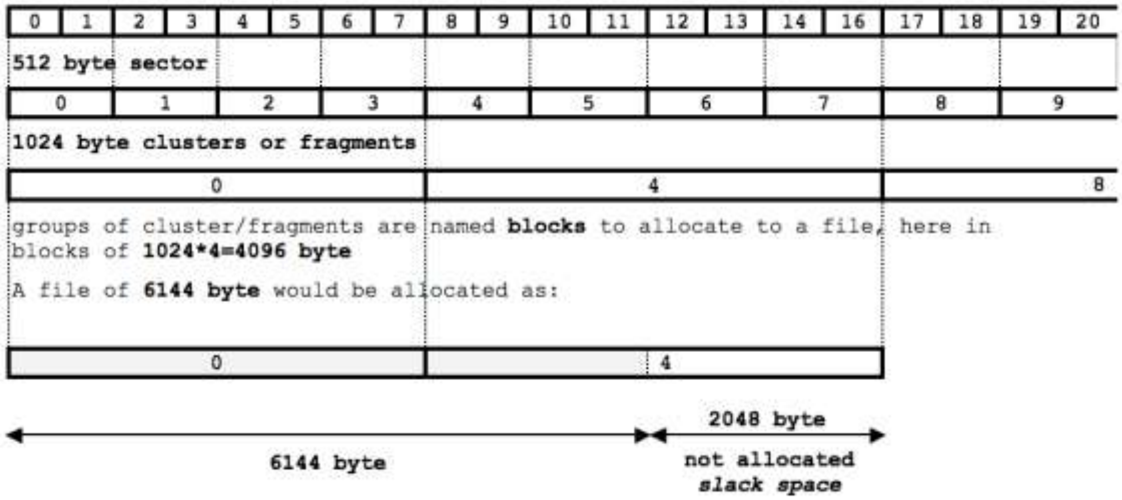


Figure 1 – a tipical file allocation

> **Note**
> Later on a method to carve data from slack space will be discussed.

It's important from a forensic perspective to know something about file system fragmentation, that typically occurs when data is not contiguously stored, due to:

- low free space;
- deletion, truncation or extension of files.

An example of fragmentation is the slack space in *figure 1*. It is worthwhile to remember, that hard disk capacity is generally much greater now than a few years ago. As a rule of thumb, it can be assumed that large hard drives are less likely to have fragmented files that the smaller ones of the past. High fragmentation eventually might be seen on large files such as email files. A drawback of fragmentation is the additional overhead due to seek time and rotation of the read/write head delays and usually can be eliminated by rearranging data storage so that related pieces are close together (*defragmentation*). New operating systems try to avoid fragmentation, with more free space available, the file system doesn't need to fill up every block. For instance, Mac OS Extended formatting (HFS Plus) avoids reusing space from deleted files as much as possible, to avoid prematurely filling small areas of recently-freed space.

> **Tip**
> Defragmentation on Mac OS X?
> Using a disk utility, repartition your HD drive, giving the strict minimum space required for the original volume, and then create a second volume with the remaining available space. This takes time, as all your data will be moved from the overall disk to the volume that is under resizing. Then when finished, get rid of the second volume you just created, giving back all available space to the original volume. It's worthwhile to do a BACKUP BEFORE!

2.1. What is a File

File is a term used in the Computer World to indicate a block of stored information (*binary digits*) such as a document in a doc file, an image in a *jpg* file or a program in an exe file. Hence, it's a up to the application to understand that block of binary digits in order to correctly show or execute the content. Files can be created, moved, modified, copied and deleted. In most cases, computer programs that are executed on the computer handle these operations, but the user of a computer can also manipulate files if necessary. Almost every computer systems use extensions in file names to help identify what they contain (*the file type*). For instance, extension consists of a dot at the end of a file name, followed by three letters to identify the type of file, see http://file-extension.net/seeker/ and http://filext.org, hence *".txt"* identifies a text file. Actually this extension had been introduced to help Operating Systems to correctly address files or rather to identify a program the file is associated with. For several reasons nowadays programs analyze the structure of a file rather than extension, and if that structure is recognized open the file, so Magic Numbers became the standard where industry has moved.

2.2. Magic Number

The term ***magic number*** has different meanings, however here we are focusing on file, hence the *magic number* is a constant used to identify a file format (Kessler, 2008). Detecting such constants in files is a simple way of distinguishing between file formats, basically every file has an header and a footer in order to get correctly recognized, for example a pdf file starts with ***"%PDF"*** and ends with ***"%EOF"*** while a jpeg image file begins with ***"0xFFD8"*** and ends with ***"0xFFD9"***. These constants are called *magic numbers*. A quick look to the unix utility "*file*" (*ftp://ftp.astrom.com/pub/file/)* clarify this concept, it can read and interpret magic numbers from files in order to determine file type.

Note

The utility "file" performs three sets of tests, in this order: file system tests, magic number tests, and language tests. The first test that succeeds causes the file type to be printed. The information identifying these files against magic numbers is read from the compiled magic file "*/usr/share/file/magic.mgc*", or "*/usr/share/file/magic*" if the compile file does not exist. Windows users might use the command "file" as part of the *cgywin* package or "TrID" (*http://mark0.net/soft-trid-e.html*), an utility designed to identify file types from their binary signatures.

Following, a look to the format of the magic file, restricted to the pdf section:

Commands

```
auditor:~ root# more /usr/share/file/magic
<*** cut ***>
# pdf:  file(1) magic for Portable Document Format
#
0       string          %PDF-           PDF document
>5      byte            x               \b, version %c
>7      byte            x               \b.%c
<*** cut ***>
auditor:~ root#
```

The first column is the offset of the byte to examine, after there is the type of the field with the value lastly the format to show. As a correct forensic analysis methodology require, be aware to use a trusted binary of the utility "*file*". Let's verify a pdf file:

Commands

```
auditor:~ root# dd if=/var/root/guide.pdf | xxd |more
0000000: 2550 4446 2d31 2e33 0a25 c4e5 f2e5 eba7  %PDF-1.3.%......
0000010: f3a0 d0c4 c60a 3220 3020 6f62 6a0a 3c3c  ......2 0 obj.<<
0000020: 202f 4c65 6e67 7468 2034 2030 2052 202f   /Length 4 0 R /
0000030: 4669 6c74 6572 202f 466c 6174 6544 6563  Filter /FlateDec
0000040: 6f64 6520 3e3e 0a73 7472 6561 6d0a 78da  ode >>.stream.x.
<*** cut ***>
0305fc0: 0a25 2545 4f46 0a                        .%%EOF.
auditor:~ root#
```

Of course file signatures can be changed, resulting in a fake file type recognition. Let's do an example, following the file "*swf2mp3*" has the "*.sh*" that let to assume it is a shell script, the "*file*" tool verify the header of the file, Unix script usually starts with a shebang "*#!*" http://en.wikipedia.org/wiki/Shebang_(Unix) followed by the path to an interpreter. M*icroprocessor architectures commonly use two different methods to store the individual bytes of multibyte numerical data in memory, this difference is referred as "byte ordering" or "endian nature", hence we would expect 0x2321*, or *0x2123* on little-endian systems as magic number)

Commands

```
auditor:~ root# file swf2mp3.sh
swf2mp3.sh: Bourne-Again shell script text executable
auditor:~ root#
```

This can be verified running an hex editor against the "*swf2mp3*" file:

Commands

```
auditor:~ root# dd if=swf2mp3.sh | xxd
0+1 records in
0+1 records out
221 bytes transferred in 0.000024 secs (9269412 bytes/sec)
0000000: 2321 2f62 696e 2f62 6173 680a 0a66 6f72  #!/bin/bash..for
0000010: 2069 2069 6e20 2428 6361 7420 6c69 7374   i in $(cat list
0000020: 6166 696c 6573 2e74 7874 293b 2064 6f0a  afiles.txt); do.
0000030: 6563 686f 2022 5374 6f20 6c61 766f 7261  echo "Sto lavora
0000040: 6e64 6f20 7375 6c20 6669 6c65 3a20 2220  ndo sul file: "
<*** cut ***>
auditor:~ root#
```

Now a change is made in the first two bytes of the previous analyzed file *"swf2mp3"* (the hex values *"23 21"* are changed in *"4D 5A"*):

```
Commands

auditor:~ root# file swf2mp3.sh
swf2mp3: MS-DOS executable (EXE)
auditor:~ root# cat swf2mp3.sh
MZ?lin/bash
for i in $(cat files_list.txt); do
echo "Im working on file: " $i
SWFExtract    -m    -o    /Users/mascalzone/Desktop/audio_files/$i.mp3
/Dati/video_security/$i.swf
done
auditor:~ root#
```

Basically what happened is that the inserted signature is the same one used for MS-DOS files. Following, the section of the magic file *"/etc/share/file/magic"* with the entry that triggered that output:

```
Commands

auditor:~ root# more /usr/share/file/magic
<*** cut ***>
0       string  MZ              MS-DOS executable (EXE)
<*** cut ***>
auditor:~ root#
```

3. Carving concepts

Data carving might be classified as basic and advanced; with basic data carving it is assumed that:

- the beginning of file is not overwritten;
- the file is not fragmented;
- the file is not compressed (i.e. NTFS compressed);

basically this type of carving is made with header and footer, while advanced data carving occurs even to fragmented files, where fragments are:

- not sequential;
- out of order;
- missing;

this type of carving relies also on internal file's structure. New operating systems try to avoid fragmentation in order to speed writing and reading of files of course unless there are conditions for which fragmentation is necessary as in the case of unavailable contiguous sectors to store the file or if data are to be appended to an existing file and no contiguous sectors are allocable. Futhermore, a malicious user might force file writing using fragmentation, in order to make it unrecoverable when deleted. The attention on advanced data carving increased because header and footer analysis do not consider the file's content, which means that sectors inserted, deleted or modified are not considered. Not only, some files have the header or *SOF* (Start Of File) but not the footer or *EOF* (End Of File); having deep knowledge of internal file's structure could result in less false positive, this is the reason of why new algorithm also relies on "internal file structure". For instance, the first sector of an office file contains a *CDH* header that must contain the hex value *FE* as the *29th* character and the value *FF* as the *30th* character, these values might be verified in order to recognize the file. Could happen that while carving a file with correct header and footer it still result unreadable because fragmentation occurred or header and footer themselves resides in different fragments.

> **Note**
>
> Header and footer are not enought to carve files because file's content is not cheked nor sector within header/footer are examined, hence deep knowledge of internal file structure is required. This means to known which sequence of bytes represent valid data object for each file type. For istance in every word doc files there are property infos (Author, Company, keywords, etc.) stored within the file structure; futher more these knowledge is also useful, for instance, to extract embedded images within word file or extract jpeg metadata. Metadata in images files are known as *Exif* (Exchangeable image file format) and can be extracted using specific tools such as in the the following example where the tool *exiftool* is run against a jpg file. Interestingly is possibile to determine the camera that took the photo, this tool is made by *Phil Harvey*

(http://www.sno.phy.queensu.ca/~phil/exiftool/), is a platform-independent Perl library plus a command-line application for reading, writing and editing meta information in image, audio and video files.

Commands

```
auditor:~ root# exiftool /Dati/exif/00019777.jpg
ExifTool Version Number         : 7.00
File Name                       : 00019777.jpg
Directory                       : /Dati/exif
File Size                       : 434 kB
File Modification Date/Time     : 2007:10:16 21:46:43
File Type                       : JPEG
MIME Type                       : image/jpeg
Exif Byte Order                 : Big-endian (Motorola)
Make                            : Hewlett-Packard
Camera Model Name               : HP PhotoSmart 318
<*** cut ***>
auditor:~ root#
```

4. Tools Testing Images

In order to test some utility in this paper the following tool testing images are used:

- data carving testing image *#11* and *#12* released on March 14, 2005 by *Nick Mikus* at *http://dftt.sourceforge.net/*:
  - image #11 (a 'raw' partition image of a FAT32 file system);
  - image #12 (a 'raw' partition image of an EXT2 file system);
- data carving DFRWS Forensics Challenge Images on line at:
  - http://dfrws.org/2006/challenge/layout.shtml;
  - http://dfrws.org/2007/challenge/layout.shtml;

For the sake of simplicity and shortness during files recovery, MD5 details are not checked, however integrity of files can be verified online at respective sites.

Note

MD5 Integrity Check

The concept of integrity is one of the most important in Forensic Analysis, however the example of this paper are not checked against md5 integrity even if is applicable on test's file here adopted.
If the reader is interested in checking the integrity, all md5 values are available on line and can be verified, just consider that some files may report incorrect md5 values, the reason is that some files do not care if additional data are appended to the end of valid file, hence md5 result different.

4.1. dftt test image #11

The image of test #11 is a FAT32 file system and is intended to test data carving tools and their ability to extract various files formats. The image contains several allocated and deleted files and the header of one jpeg file was modified (to show the importance of ignoring corrupted files). All files are random files created from scratch. This image was created from a USB thumb-drive that was wiped and formatted using the *mkfs.vfat* program. The FAT boot sector has been corrupted so that the image cannot be mounted and therefore data carving methods must be used to extract the files. In the following table details about files included in the image are reported:

#	Name	Size	Note	Sectors
1	2003_document.doc	19968	A Valid DOC file	(0-38) 281-320
2	enterprise.wav	318895	A valid WAV file	(0-622) 16021-16644
3	haxor2.jpg	24367	An invalid JPEG with only 1 header byte corrupted. This byte is located at offset 19 within the file.	(0-47) 16645-16692
4	holly.xls	23040	A valid XLS file	(0-44) 16693-16738
5	lin_1.2.pdf	1399508	A linearized PDF	(0-2733) 16741-19475
6	nlin_14.pdf	122434	A non linearized PDF	(0-239) 19477-19716
7	paul.jpg	29885	A valid jpeg	(0-58) 19717-19776
8	pumpkin.jpg	444314	A valid EXIF jpeg	(0-867) 19777-20644
9	shark.jpg	99298	A valid JPEG	(0-193) 20645-20839
10	sm1.gif	5498	A valid GIF	(0-10) 20841-20852
11	surf.mov	550653	A valid MOV	(0-1075) 20853-21928
12	surf.wmv	1036994	A valid WMV	(0-2025) 21929-23955
13	test.ppt	11264	A deleted PPT	(0-21) 23957-23978
14	wword60t.zip	78899	A valid ZIP	(0-154) 23981-24135
15	domopers.wmv	8037267	A deleted wmv	(0-15697) 321-16018

4.2. dftt test image #12

The image of test #12 is an EXT2 file system and is intended to test data carving tools for indirect block detection and removal. With large files, EXT2 allocates blocks (called indirect blocks) to store file metadata and the blocks are frequently allocated in between blocks that contain file content. Therefore, the file becomes fragmented and a basic carving tool may include the indirect block in the carved file.

This file system image contains several allocated and deleted files, none of which have been modified. This image was created from a usb thumb drive that was wiped clean and formatted using the *mkfs.ext2* program. The super block has been corrupted so that the image cannot be mounted and therefore data carving methods must be used to extract the files. In the following table details about files included in the image, the sectors marked as *"(IND)"* and *"(DIND)"* represent the indirect and double indirect block pointer locations:

#	Name	Size	Note	Sectors
1	haxor2.bmp	163878	A deleted BMP	(0-22):5162-5184,(IND):5186,(24-320):5188-5484
2	jimmy.doc	12800	A deleted DOC	(0-22):5486-5508,(IND):5510,(24):5512
3	jn.jpg	28949	A valid JPG	(0-22):5514-5536,(IND):5538,(24-56):5540-5572
4	lin_test.pdf	26618	A valid PDF	(0-22):5574-5596,(IND):5598,(24-50):5600-5626
5	main_dive.jpg	8463	A valid jpeg	(0-16):5628-5644
6	n_lin_ss.pdf	734652	A valid pdf	(0-22):5646-5668,(IND):5670,(24-534):5672-6182, (DIND):6184,(IND):6186,(536-1046):6188-6698, (IND):6700,(1048-1434)6702-7088
7	blogo.gif	18663	A valid gif	(0-22):5122-5144,(IND):5146,(24-36):5148-5160
8	sherry.jpg	133249	A valid JPEG	(0-22):7090-7112,(IND):7114,(24-260):7116-7352
9	stats.xls	15360	A valid XLS	(0-22):7354-7376,(IND):7378,(24-28):7380-7384
10	test.ppt	17408	A valid PPT	(0-22):7386-7408,(IND):7410,(24-32):7412-7420

5. DFRWS Forensics Challenge Images

The importance of well done data carving tools is highlighted by two challenges released by the Digital Forensic Research Workshop at http://dfrws.org/. Basically two years in a row had been dedicated to data carving. The 2006 challenge was focused on carving basic file types in basic scenarios with the goal to develop new tools and techniques to carve files using more internal structure than only the header and footer values. The 2007 challenge introduced more file types and more complex fragmentation scenarios. The goal was to design and develop automated file carving algorithms with high true positive and low false positive rates. This challenge was much more difficult than the previous and the workgroup received feedback at the DFRWS 2007 conference that many did not even attempt this challenge because it seemed too daunting, hence in the following section only this challenge will be discussed. Many of the scenarios in the challenge involved fragmented files where fragments were sequential, out of order, or missing. Existing tools, as later prooved, could not handle these scenarios and new techniques had to be developed.

5.1. 2006 Challenge

The data set for this challenge at http://dfrws.org/2006/challenge/ contains 32 files (not including the embedded files, such as pictures in Word documents or the files inside of ZIP files). The 32 files were used to create 22 different scenarios, which are described online. Each scenario was designed to test a specific situation that might occur in a real file system. For example, there were different scenarios for fragmented jpeg files with text in between and with random data in between. The data in between each scenario is random, the image file was initialized by creating a 50MB file with random data (using */dev/urandom*).

5.2. 2007 Challenge

The data set for this challenge at

http://dfrws.org/2007/challenge/ is a 330MB raw file without file system data, it contains many files and file fragments. The file types used in the DFRWS 2006 carving challenge was also included in this challenge (jpeg, zip, html, text and microsoft office) along with more multimedia, document, and e-mail formats. Example new formats include, but not limited to, mp3, mpg, wmv, pdf, and executables. All files in the image were distributed using their original copyrights. Interestingly, manually use of data from the Internet to find a full file from a fragment was not allowed. This meant that finding a fragment in the challenge data, manually searching the Internet for the full file, and then finding the remaining file fragments in the disk image was not graded, while automated techniques that used the Internet were considered. For example, find a file fragment in the data, search the Internet for that fragment, and then search the data for the remaining fragments was considered valid. In order to better understand the fragmentation issue, following a graphical view of some files fragmentation introduced within the challenge image is showed:

intertwined files, A is fragmented and B sits between

a file of 2 fragments in sequential order

a file of 2 fragments in non sequential order

a file of 3 fragments in sequential order, last fragment with EOF is missing

a file of 3 fragment in sequential order with fragment #2 missing

6. Unallocated data recovery and slack space

Sometimes, where it is needed to recover deleted data, might be useful to run the tool *dls* against the system device before carves data, this in order to extract all information from the unallocated data. *dls* is part of the Sleuth Kit (http://www.sleuthkit.org) a collection of file system analysis tools (derived from TCT http://www.coronertoolkit.org and *TCTUTILs*), written by *Brian Carrier*. The tool was called *unrm* in *TCT* and was well known as the previous step before running the *lazarus* tool, it has been renamed in order to conform to the naming convention of the *Sleuth Kit*. In the following example */dev/disk1* is *1 GB* of usb key with about *25,5 MB* of data on it, *dls_usb_key.img* returned about *886,6 MB* (the free space) and took about *12 min*. This tool is also used to extract slack space (data from the end of the file to the end of the cluster – figure 1) of FAT and NTFS images.

```
Commands
auditor:~ root# date; dls /dev/disk1  > dls_usb_key.img; date
Fri Oct 19 20:44:17 CEST 2007
Fri Oct 19 20:55:53 CEST 2007
auditor:~ root# df -k /dev/disk1
Filesystem 1K-blocks  Used  Avail Capacity  Mounted on
/dev/disk1   1014256 26096 988160     3%    /Volumes/USBKEY
auditor:~ root# ls -l dls_usb_key.img
-rw-r--r--   1 auditor  admin  929648640 Oct 19 20:55 dls_usb_key.img
```

Note

If interested in tools for data recovery as part of the sleuth kit, could be interesting evaluate *comeforth* an add-on of the *Sleuth Kit*, which parse raw filesystem blocks, or block image data produced by *dls*. This tool (inspired by *lazarus*) seems to provide a bit more flexibility for processing very large data sets, basically *comeforth* is really useful in recovering data if a file is not stored in sequence.
After scanning, blocks that have been saved can be viewed, and based on their contents files can be reassembled from various other blocks. An auto-assemble feature is provided which can reassemble a complete file in many cases, knowing only the first block in the file (only for ext2/ext3 filesystems).

It is worthwhile to remember the Autopsy (http://www.sleuthkit.org/autopsy/) a forensic browser GUI to the command line tools of The Sleuth Kit that allow to investigate the file system and volumes of a computer.

7. Tools

Of course there are lots of tools to carve files, and new ones are coming to the community. In this paper just the most well known tools are evaluated, while reading goes on, will be highlithed the necessity to develop an unique program joining all the features and power of several tools available to users.

The reader might also be interested in others tools such as those listed online at:

http://www.forensicswiki.org/index.php?title=Tools:Data_Recovery#Carving.

7.1. Foremost

Foremost is a well-known tool, originally developed at the US AirForce (Developed by Kris Kendall and Jesse Kornblum of the USA Air Force Office of Special Investigations, http://foremost.sourceforge.net/) it works on image files, such as those generated by *dd*, *Safeback*, *Encase*, etc. or directly on a drive. A configuration file can specify the headers and footers or can use command line switches to specify built-in file types. These built-in types look at the data structures of a given file format allowing for a more reliable and faster recovery. In the following example the *pdf* format is showed, basically there is the type of the file "*pdf*", the "*y*" that stands for 'yes' and means that the header and footer are case sensitive, *5000000* is the maximum size of the file within the footer is searched before give up. We already discussed about the other two fields (header and footer of the file type):

Commands

```
auditor:~ root# more /etc/foremost.conf
<*** output cut ***>
#---------------------------------------------------------------------
# ADOBE PDF   (NOTE THIS FORMAT HAS A BUILTIN EXTRACTION FUNCTION)*
#---------------------------------------------------------------------
#
#       pdf     y       5000000 %PDF-  %EOF
<*** cut ***>
auditor:~ root#
```

Note

Note the foremost configuration file is provided to support formats that don't have built-in extraction functions. If the format is built-in to foremost simply run foremost with -t <suffix> and provide the format you wish to extract.

To get the list of command lines switches, just type:

```
Commands
auditor:~ root# foremost -h
foremost version 1.5 by Jesse Kornblum, Kris Kendall, and Nick Mikus.
$ foremost [-v|-V|-h|-T|-Q|-q|-a|-w-d] [-t <type>] [-s <blocks>] [-k
<size>]
        [-b <size>] [-c <file>] [-o <dir>] [-i <file]

-V  - display copyright information and exit
-t  - specify file type.  (-t jpeg,pdf ...)
-d  - turn on indirect block detection (for UNIX file-systems)
-i  - specify input file (default is stdin)
-a  - Write all headers, perform no error detection (corrupted files)
-w  - Only write the audit file, do not write any detected files to
the disk
-o  - set output directory (defaults to output)
-c  - set configuration file to use (defaults to foremost.conf)
-q  - enables quick mode. Search are performed on 512 byte
boundaries.
-Q  - enables quiet mode. Suppress output messages.
-v  - verbose mode. Logs all messages to screen
auditor:~ root#
```

From the above output author *Nick Mikus* has been added to the original authors, *Mikus* while working on his master's thesis (Mikus, 2005) improved the tool adding useful module regarding metadata information, specific to the files format referred as internal file structure validators, along with integration of file system specific techniques. The output directory (specified with *-o*) will contain the result file *audit.txt*. Following, the file *audit.txt* against the images test file *#11*:

Commands

```
Foremost version 1.5 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File

Foremost started at Tue Oct 16 21:46:42 2007
Invocation: foremost -v -c /etc/foremost.conf -o carving11/ -i
/Dati/11-carve-fat/11-carve-fat.dd
Output directory: /Users/auditor/carving11
Configuration file: /etc/foremost.conf
------------------------------------------------------------------
File: /Dati/11-carve-fat/11-carve-fat.dd
Start: Tue Oct 16 21:46:42 2007
Length: Unknown

Num    Name (bs=512)           Size  File Offset       Comment

0:    00019717.jpg           29 KB          10095104
1:    00019777.jpg          433 KB          10125824
2:    00020645.jpg           96 KB          10570240
3:    00020841.gif            5 KB          10670592  (88 x 31)
4:    00000321.wmv            7 MB            164352
5:    00021929.wmv         1012 KB          11227648
6:    00020853.mov          537 KB          10676740
7:    00016021.wav          311 KB           8202752
8:    00000281.doc           20 KB            143872
9:    00016693.xls           24 KB           8546816
10:   00023957.ppt           13 KB          12265984
11:   00023981.zip           77 KB          12278272
12:   00016741.pdf            1 MB           8571392 (PDF is
                                                      Linearized)
13:   00019477.pdf          119 KB           9972224
Finish: Tue Oct 16 21:46:45 2007

14 FILES EXTRACTED

jpg:= 3
gif:= 1
wmv:= 2
mov:= 1
rif:= 1
ole:= 3
zip:= 1
pdf:= 2
------------------------------------------------------------------

Foremost finished at Tue Oct 16 21:46:45 2007
```

What is to point out here is that *14 files* are carved while in the table they are *15* of them, the missing file is the *#3 haxor2.jpg*, an invalid jpeg file. Following, the *audit.txt* file against test file *#12* is reported. In this second data recovery attempt *foremost* is executed with the switch ***-d*** in order to turn on indirect block detection without this switch the tool carves only *7 files*.

Commands

```
Foremost version 1.5 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File

Foremost started at Tue Oct 16 22:22:49 2007
Invocation: foremost -v -d -c /etc/foremost.conf -o carving12/ -i
/Dati/12-carve-ext2/12-carve-ext2.dd
Output directory: /Users/auditor/carving12
Configuration file: /etc/foremost.conf
------------------------------------------------------------------
File: /Dati/12-carve-ext2/12-carve-ext2.dd
Start: Tue Oct 16 22:22:49 2007
Length: Unknown

Num   Name (bs=512)      Size File Offset       Comment

0:    00005514.jpg       28 KB      2823168      (IND BLK bs:=1024)
1:    00005628.jpg        8 KB      2881536
2:    00007090.jpg      130 KB      3630080      (IND BLK bs:=1024)
3:    00005122.gif       18 KB      2622464      (IND BLK bs:=1024)
                                                 (620 x 802)
4:    00005161.bmp      160 KB      2642940      (IND BLK bs:=1024)
                                                 (400 x 407)
5:    00005486.doc       14 KB      2808832      (IND BLK bs:=1024)
6:    00007354.xls       17 KB      3765248      (IND BLK bs:=1024)
7:    00007386.ppt       19 KB      3781632      (IND BLK bs:=1024)
8:    00005574.pdf       25 KB      2853888      (IND BLK bs:=1024)(PDF
                                                 is Linearized)
9:    00005646.pdf      720 KB       2890752     (IND BLK bs:=1024)

Finish: Tue Oct 16 22:22:57 2007

10 FILES EXTRACTED

jpg:= 3
gif:= 1
bmp:= 1
ole:= 3
pdf:= 2
------------------------------------------------------------------

Foremost finished at Tue Oct 16 22:22:57 2007
```

7.2. Scalpel

Scalpel at *http://www.digitalforensicssolutions.com/Scalpel/* is a complete rewrite of *foremost 0.69* done by *Golden G. Richard III*, to enhance performance and decrease memory usage. It is a fast and filesystem-independent file carver that reads a database of header and footer definitions and extracts matching files from a set of image files or raw device files. Interestingly as of version *1.53*, is supported on *Mac OS X* and as of version *1.54*, supports live carving of block devices under *Mac OS X*. In particular, maximum file carve size under *Foremost 0.69* is *4GB* while in the current version of *Scalpel*, it's *16EB* (*16 exabytes*). To get the list of command lines switches, just type:

Commands

```
auditor:~ root# scapel -h
Scalpel version 1.60
Written by Golden G. Richard III, based on Foremost 0.69.
Carves files from a disk image based on file headers and footers.

Usage: scalpel [-b] [-c <config file>] [-d] [-h|V] [-i <file>]
                  [-m blocksize] [-n] [-o <outputdir>] [-O num] [-q
            clustersize] [-r] [-s num] [-t <blockmap file>] [-u] [-v]
                  <imgfile> [<imgfile>] ...

-b  Carve files even if defined footers aren't discovered within
    maximum carve size for file type [foremost 0.69 compat mode].
-c  Choose configuration file.
-d  Generate header/footer database; will bypass certain
    optimizations
    and discover all footers, so performance suffers.  Doesn't affect
    the set of files carved.  **EXPERIMENTAL**
-h  Print this help message and exit.
-i  Read names of disk images from specified file.
-m  Generate/update carve coverage blockmap file.  The first 32bit
    unsigned int in the file identifies the block size. Thereafter
    each 32bit unsigned int entry in the blockmap file corresponds
    to one block in the image file.  Each entry counts how many
    carved files contain this block. Requires more memory and
    disk.  **EXPERIMENTAL**
-n  Don't add extensions to extracted files.
-o  Set output directory for carved files.
-O  Don't organize carved files by type. Default is to organize
    carved files into subdirectories.
-p  Perform image file preview; audit log indicates which files
    would have been carved, but no files are actually carved.
```

```
-q  Carve only when header is cluster-aligned.
-r  Find only first of overlapping headers/footers [foremost 0.69
    compat mode].
-s  Skip n bytes in each disk image before carving.
-t  Set directory for coverage blockmap.  **EXPERIMENTAL**
-u  Use carve coverage blockmap when carving.  Carve only sections
    of the image whose entries in the blockmap are 0.  These areas
    are treated as contiguous regions.  **EXPERIMENTAL**
-V  Print copyright information and exit.
-v  Verbose mode.
```

The output directory (specified with ***-o***) will contain results. Following Scalpel run against image #11, the switch **-b** stands for carve files even if defined footers aren't discovered within maximum carve size (foremost 0.69 compat mode). **–c** in order to specify the configuration file (the default one)

Commands

```
Scalpel version 1.60 audit file
Started at Tue Oct 30 16:59:54 2007
Command line:
scalpel -b -c scalpel.conf -o /Users/auditor/test-11 -O /Dati/11-
carve-fat /11-carve-fat.dd

Output directory: /Users/auditor/test-11
Configuration file: /Users/auditor/carvers/scalpel-1.60/scalpel.conf

Opening target "/Dati/11-carve-fat/11-carve-fat.dd"

The following files were carved:
File           Start           Chop        Length      Extracted From
00000002.jpg  10095104        NO           29885      11-carve-fat.dd
00000001.jpg  8522240         NO           24367      11-carve-fat.dd
00000008.doc  143872          NO           8402944    11-carve-fat.dd
00000011.doc  143872          YES         10000000    11-carve-fat.dd
00000014.pdf  8571392         NO           1399508    11-carve-fat.dd
00000015.pdf  8571392         NO           1523266    11-carve-fat.dd
00000016.pdf  9972224         NO            122434    11-carve-fat.dd
00000009.doc  8546816         NO           3719168    11-carve-fat.dd
00000012.doc  8546816         YES         10000000    11-carve-fat.dd
00000005.jpg  10574693        NO              2655    11-carve-fat.dd
00000004.jpg  10570636        NO              2655    11-carve-fat.dd
00000003.jpg  10570240        NO              3051    11-carve-fat.dd
00000000.gif  10670592        NO              5498    11-carve-fat.dd
00000017.zip  12278272        NO          10000000    11-carve-fat.dd
00000013.doc  12265984        YES         10000000    11-carve-fat.dd
00000010.doc  12265984        NO          10000000    11-carve-fat.dd
00000006.png  10149442        NO          20000000    11-carve-fat.dd
00000007.mpg  5329297         NO          50000000    11-carve-fat.dd


Completed at Tue Oct 30 17:00:03 2007
```

The same test run against image #12:

```
Commands
Scalpel version 1.60 audit file
Started at Tue Oct 30 17:00:46 2007
Command line:
scalpel -b -c scalpel.conf -o /Users/auditor/test-12 -O /Dati/12-
carve-ext2/12-carve-ext2.dd

Output directory: /Users/auditor/test-12
Configuration file: /Users/auditor/carvers/scalpel-1.60/scalpel.conf

Opening target "/Dati/12-carve-ext2/12-carve-ext2.dd"

The following files were carved:
File           Start          Chop        Length      Extracted From
00000006.doc  3765248         NO           16384      12-carve-ext2.dd
00000005.doc  2808832         NO           956416     12-carve-ext2.dd
00000004.jpg  3632218         NO           4905       12-carve-ext2.dd
00000003.jpg  3630080         NO           7043       12-carve-ext2.dd
00000002.jpg  2881536         NO           8463       12-carve-ext2.dd
00000001.jpg  2823168         NO           29973      12-carve-ext2.dd
00000000.gif  2622464         NO           19687      12-carve-ext2.dd
00000011.pdf  2853888         NO           27642      12-carve-ext2.dd
00000012.pdf  2853888         NO           775612     12-carve-ext2.dd
00000013.pdf  2890752         NO           738748     12-carve-ext2.dd
00000010.doc  3781632         YES         10000000    12-carve-ext2.dd
00000009.doc  3765248         YES         10000000    12-carve-ext2.dd
00000008.doc  2808832         YES         10000000    12-carve-ext2.dd
00000007.doc  3781632         NO          10000000    12-carve-ext2.dd


Completed at Tue Oct 30 17:00:58 2007
```

As matter of fact seems that Scalpel findings are not accurate as would expect, thus a comparison table had been created in order to highlight the differences with tests run using Foremost.

7.3. Foremost/Scalpel Comparison Table

A quick comparison of Foremost and Scalpel is here analyzed in order to point out differences:

#11	File Name	Foremost	Scalpel	Note
1	2003_document.doc	00000281.doc	00000008.doc 00000011.doc	Valid – Scalpel returned 2 different files in size but same content
2	enterprise.wav	00016021.wav	-	Valid – Scalpel Missing
3	haxor2.jpg	-	-	Not recovered!
4	holly.xls	00016693.xls	00000009.doc 00000012.doc	Valid - Scalpel returned 2 different doc files in size but same content as the correct xls
5	lin_1.2.pdf	00016741.pdf	00000014.pdf	Valid
6	nlin_14.pdf	00019477.pdf	00000015.pdf 00000016.pdf	Valid - Scalpel returned 2 different files in size but same content
7	paul.jpg	00019717.jpg	00000002.jpg	Valid
8	pumpkin.jpg	00019777.jpg	-	Valid – Scalpel Missing
9	shark.jpg	00020645.jpg	00000004.jpg 00000005.jpg	Valid; image size better with Foremost and Scalpel returned 2 different files in size but same content
10	sm1.gif	00020841.jpg	00000000.gif	Valid
11	surf.mov	00020853.mov	-	Valid – Scalpel Missing
12	surf.wmv	00021929.wmv	-	Valid – Scalpel Missing
13	test.ppt	00023957.ppt	-	Valid – Scalpel Missing
14	wword60t.zip	00023981.zip	00000017.zip	Valid
15	domopers.wmv	00000321.wmv	-	Valid – Scalpel Missing

Foremost started at 21:46:42 and finished at 21:46:45, took 3 sec. Scalpel started at 16:59:54 and finished at 17:00:03, took 9 sec.

#12	File Name	Foremost	Scalpel	Note
1	haxor2.bmp	00005161.bmp	-	Valid – Scalpel Missing
2	jimmy.doc	00005486.doc	-	Valid – Scalpel Missing

3	jn.jpg	00005514.jpg	00000001.jpg	Valid but incomplete with Scalpel
4	lin_test.pdf	00005574.pdf	00000011.pdf	Valid
5	main_dive.jpg	00005628.jpg	00000002.jpg	Valid
6	n_lin_ss.pdf	00005646.pdf	00000012.pdf 00000013.pdf	Valid - Scalpel returned 2 different files in size but same content
7	blogo.gif	00005122.gif	00000000.jpg	Valid but incomplete with Scalpel
8	sherry.jpg	00007090.jpg	00000004.jpg	Valid but image size better with Foremost
9	stats.xls	00007354.xls	-	Valid – Scalpel Missing
10	test.ppt	00007386.ppt	-	Valid – Scalpel Missing

Foremost started at 22:22:49 and finished at 22:22:57 and took 8 sec, Scalpel started at 17:00:46 and finished at 17:00:58, took 12 sec. This comparison had been sent to the scalpel's author in order to get a feedback about the tool behaviour. However at first glance, the reader can conclude that both tools had been run with a default installation and configuration and Foremost had been more reliable.

8. Foremost against DFWS 2007 image file

Considering that Foremost performed well in the previous tests, it will be run against the DFRWS 2007 image file, just in order to show the importance of the challenge's goal to develop and/or improve tools for this purposes. However Foremost during the test reported a "*segmentation fault*" when run against this image (the author had been contacted in order to get a feedback). Hence to continue test avoiding the segmentation fault it had been run only against *jpeg* files (using the command switch **-t** jpg). Within the image file the *jpgs* are arranged as:

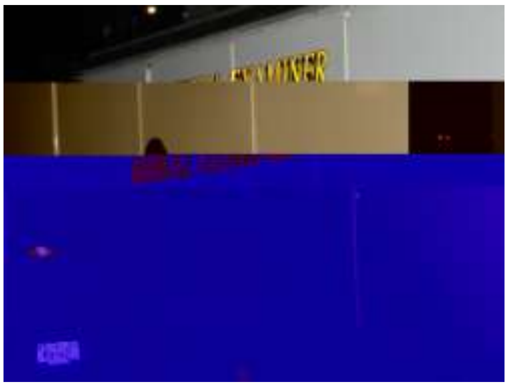
Name	***Idx***	***SectorOffset***	***ImageSectorOffset***	***Scenario***	***Description***
1.jpg	0	000000-000295	648613-648908	2	a JPG file not fragmented

2.jpg	0 1	000000-000300 000301-002149	057145-057445 057504-059352	15	a JPG with filler in between with fragments in sequential order
3.jpg	0 1	000000-001347 001348-003964	089029-090376 091052-093668	16	a JPG file intertwined with fragments in sequential order
4.jpg	0 1 2	000000-000674 000675-000785 000786-001571	090377-091051 093669-093779 099664-100449		
5.jpg	0 1	000000-005883 005884-007742	093780-099663 100450-102308		
6.jpg	0 1	000000-005310 005311-006035	336372-341682 342827-343551	17	a JPG file intertwined with fragments in sequential order
7.jpg	0 1	000000-000014 000015-000514	341683-341697 343714-344213		
8.jpg	0 1	000000-001128 001129-002091	341698-342826 344214-345176		
9.jpg	0 1	000000-000161 000162-000508	343552-343713 345177-345523		
10.jpg	1 0	000794-000956 000000-000793	334933-335095 335096-335889	38	a 2 frags JPG file (2, 1) with fragments in non-sequential order
11.jpg	2 0 1	003628-004711 000000-000705 000706-003627	519316-520399 520400-521105 521106-524027	39	a 3 frags JPG file (3, 1, 2) with fragments in non-sequential order
12.jpg	2 1 0	000251-000292 000243-000250 000000-000242	087538-087579 087677-087684 087716-087958	40	a 3 frags JPG file (3, 2, 1) with fragments in non-sequential order
13.jpg	0 2 1	000000-000389 001038-001116 000390-001037	444785-445174 445175-445253 445254-445901	41	a 3 frags JPG file (1, 3, 2) with fragments in non-sequential order
14.jpg	0 1	000000-000484 000485-000538	070034-070518 not used	60	a 2 frags JPG file (1) with missing end
15.jpg	0 2 1 3	000000-000375 000577-000876 000376-000576 000877-001790	649204-649579 649580-649879 649880-650080 650081-650994	42	a 4 frags JPG file (1, 3, 2, 4) with fragments in non-sequential order
16.jpg	0 2 1	000000-000254 000619-000982 000255-000618	350010-350264 350265-350628 not used	61	a 3 frags JPG file (1, 3) with missing middle
17.jpg	0 1 2	000000-000243 000244-000488 000489-000679	254789-255032 255165-255409 not used	62	a 3 frags JPG file (1, 2) with missing end
18.jpg	2 0 1	000425-000634 000000-000183 000184-000424	445902-446111 446112-446295 not used	63	a 3 frags JPG (3, 1) file with missing middle and fragments in non-sequential order

Following foremost against the image mentioned above:

```
Commands
Foremost version 1.5 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File

Foremost started at Wed Mar 26 18:07:55 2008
Invocation:
foremost -t jpg -c /etc/foremost.conf -o foremost_dfrws_2007/ -i
/Users/auditor/Desktop/dfrws-2007-challenge.img

Output directory: /Users/auditor/foremost_dfrws_2007
Configuration file: /private/etc/foremost.conf
------------------------------------------------------------------
File: /Users/auditor/Desktop/dfrws-2007-challenge.img
Start: Wed Mar 26 18:07:55 2008
Length: Unknown

Num   Name (bs=512)            Size       File Offset  Comment

0:    00057145.jpg             1 MB         29258240
1:    00087716.jpg           166 KB         44910592
2:    00089029.jpg           683 KB         45582848
3:    00093780.jpg             3 MB         48015360
4:    00254789.jpg           434 KB        130451968
5:    00335096.jpg           406 KB        171569152
6:    00341683.jpg            16 KB        174941696
7:    00343552.jpg           330 KB        175898624
8:    00350010.jpg           309 KB        179205120
9:    00446112.jpg           166 KB        228409344
10:   00520400.jpg             1 MB        266444800
11:   00591405.jpg           157 KB        302799747
12:   00591723.jpg            68 KB        302962390
13:   00591867.jpg            79 KB        303036049
14:   00592032.jpg            62 KB        303120770
15:   00592159.jpg            83 KB        303185820
16:   00592330.jpg            41 KB        303273404
17:   00592413.jpg            51 KB        303315850
18:   00592521.jpg            65 KB        303370861
19:   00592817.jpg            22 KB        303522618
20:   00592866.jpg            58 KB        303547507
21:   00592982.jpg            23 KB        303607104
22:   00593032.jpg            21 KB        303632516
23:   00648613.jpg           147 KB        332089856
24:   00649204.jpg           895 KB        332392448
Finish: Wed Mar 26 18:07:57 2008

25 FILES EXTRACTED

jpg:= 25
------------------------------------------------------------------

Foremost finished at Wed Mar 26 18:07:57 2008
```

The total number of extracted files is 25, the reason is that even a single file fragment is considered a valid *jpeg* file and as matter of fact all files are not corrupted and is possible to open them with an image software editor, for instance the Foremost carved file #24 (*00649204.jpg*) is the file #15 within the image, a four fragmented file with fragments in non-sequential order. Following the two jpg files are showed in order to be compared:



Foremost – jpg #24 **dfrws original file** – jpg #15

There were five submissions to the DFRWS 2007 Forensics Challenge, at the time of challenge, existing tools could not handle the presented scenarios and new techniques had to be developed. To summarize the result, the winner developed dedicated validators for pdf, zip, mime, html and mpeg and even though he did not focus on image and office file formats, his results still ended up very high, with the lowest false positive score. The second place, authors of the challenge developed several new tools for specific file types and the structure-based tool for mp3 files scored better than the other submissions; the third place author used a combination of techniques and all the pdf files were found with a tool that scans the image for PDF\Title headers and then searches the web. In fact, this is the only way that fragmented files were recovered, authors of the fourth place used an approach which combines file structure details, content analysis, and block-based carving they released a tool called Revit07 a file carver that uses a combination of file structure and content based carving techniques. The tool is available along with a very well done Master Thesis (Kloet, 2007); this work shows promise.

```
Commands
auditor@audit revit07-20070804 $ revit -h
revit 20070804

Usage: revit [-ABeFhqvV] [-b block_size] [-c configuration_file] [-t
target_directory] data_file
      -A: write the analysis log (analysis.log)
      -b: specify the block size (default is: 512)
      -B: write the buffer characteristics log (buffer.log)
      -c: specify the configuration file (default is:
          ./file_types.conf)
      -e: also revive embedded files
      -F: enable search for fragments
      -h: shows this usage information
      -q: be quiet, do not print status indicator
      -t: specify the target directory (default is: revived)
      -v: verbose output to stderr
      -V: print version

auditor@audit revit07-20070804 $ revit -F -c
/usr/local/etc/file_types.conf
/Users/auditor/data_carving/images/dfrws-2007-challenge.img
```

Basically the results is compared with carving done by Foremost, some *jpg* files are missing in Foremost while some others are missing in Revit07, hence once again a situation where is better to relie on several tools and join the results, especially in the delicated field of forensic analysis.

9. Other "Carving"

Data can be carved not only from a data block but also from network traffic or from RAM, following a brief introduction of some tools able to carves data from other sources.

9.1. tcpxtract

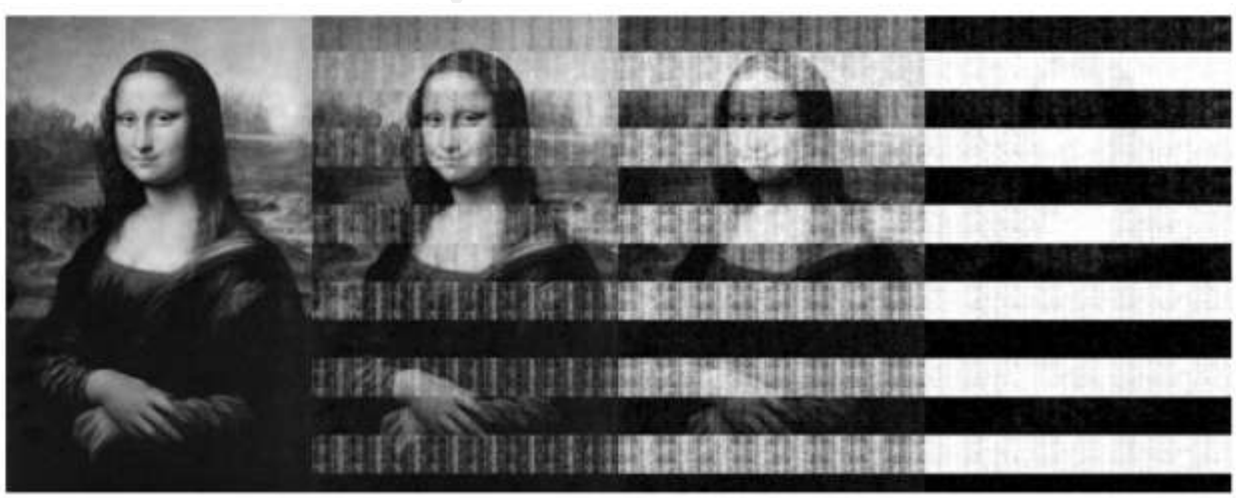
Tcpxtract at http://tcpxtract.sourceforge.net/ is a freeware tool written by *Nick Harbour* for extracting files from network traffic, based on file signatures (headers and footers) it uses the same techniques used by *foremost*, but is specifically for the application of intercepting files transmitted across a network. *tcpxtract* supports 26 popular file formats out-of-the-box, new formats can be added by simply editing its configuration file. With a quick conversion, is possible to use *foremost's* configuration file. The tool uses *libpcap*, a popular portable and stable library for network data capture and moreover can be used against a live network or a *tcpdump* formatted capture file.

9.2. chaosreader

Chaosreader at http://chaosreader.sourceforge.net/ is a freeware tool written by *Brendan Gregg*, it can trace *TCP/UDP/etc.* sessions and fetch application data from *tcpdump* or *snoop* logs. It fetches *telnet* sessions, *FTP* files, *HTTP* transfers (*HTML, GIF, JPEG, etc.*), *SMTP* emails, etc. from the captured network traffic. As output, it creates an *html* index file that links to all the session details, including realtime replay programs for *telnet*, *rlogin* or *IRC* sessions, hence is possible to play them back in realtime (or even different speeds). *Chaosreader* can also run in standalone mode, where it invokes *tcpdump* or *snoop* (if they are available) to create the log files and then processes them. There are several examples output at *http://www.brendangregg.com/chaosreader.html*, interestingly there are also perl programs such as the one to replay the displayed text from a session or the one that redisplays the VNC session.

9.3. msramdmp

Msramdmp at http://www.mcgrewsecurity.com/projects/msramdmp/ is a freeware tool written by Wesley McGrew, based on a paper titled "Lest We Remember: Cold Boot Attacks on Encryption Keys" (Princeton, 2008), as most just assume that, since RAM is a volatile storage, it is erased when power is removed. Well, the research just demonstrate that this assumption might be incorrect! however going beyond this goal, the tool can be used to carves out data from memory (this result can be obtained with the well known tool *dd* against the mem device). Lots of videos and infos are available by the researchers, as the following one where they loaded an image into memory, then cut power for varying lengths of time. After 5 seconds (left), the image was indistinguishable from the original. It became gradually more degraded, as showed after 30 seconds, 60 seconds, and 5 minutes. The horizontal bars result from the design of the tested memory chip, which represented some "1" bits by the presence of charge and some by the absence of charge.



5 sec	30 sec	60 sec	300 sec

10. Conclusion

Of course this paper could have been more detailed; however the goal was to introduce the reader within the data carving concepts of the forensic system analysis field. Surely, all references reported in this paper will help to get more detail about this topic. In the end, people who wants to be actively involved in the theory and technics of data carving seriously needs to delve into File System of the most spread Operating Systems along with a deep internal file structure knownledge. Lessons learned are related on how to perform data carving or rather which tools are available out there. A list of testing image files to practice had been overviewed, take into account to use these images in order to validate improvement of the tools here showed or new ones that might be developed in the future.

11. References

Garfinkel, Simson File Carving. from forensicswiki Web site: http://www.forensicswiki.org/wiki/Carving

Carrier, Brian (2005). *File System Forensic Analysis*. Addison Wesley.

Kessler, Gary C. (10/2/2008). File Signature Table. Web site: http://www.garykessler.net/library/file_sigs.html

Mikus, N. (2005). Master Thesis *An Analysis of Disc Carving Techniques*. Web site: http://handle.dtic.mil/100.2/ADA432468

Kloet, S.J.J. (2007). Master Thesis *Measuring and Improving the quality of File Carving Methods*. Web site: http://www.uitwisselplatform.nl/projects/revit/

Garfinkel, S.L. (2007). *Carving contiguous and fragmented files with fast object validation* Elsevier, Digital Investigation

(DFRWS) Digital Forensic Research Workshop. Web site: http://dfrws.org/

Princeton University, (2008). *Lest We Remember: Cold Boot Attacks on Encryption Keys*. from Center for Information Technology Policy Web site: http://citp.princeton.edu/memory/