Facultad de Ciencias

Recuperación de información en sistemas de ficheros

File recovering in filesystems

Trabajo de fin de grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Pablo Sáez Alonso

Director: Domingo Gómez Pérez

junio de 2024

Agradecimientos

Este trabajo fin de grado es parte del proyecto CÁTEDRA UNIVERSIDAD DE CANTABRIA-INCIBE DE NUEVOS RETOS EN CIBERSEGURIDAD, financiado por la Unión Europea NextGeneration-EU, Plan de Recuperación, Transformación y Resiliencia, a través de INCIBE.

Recuperación de información en sistemas de ficheros

Resumen

palabras clave: Análisis Forense, FAT

El análisis forense de ordenadores es una disciplina de la informática que se ocupa de asuntos relacionados con la aplicación de las leyes en el ámbito público. Este análisis, que se realiza en ordenadores y otros dispositivos informáticos, tiene fines jurídicos y es fundamental garantizar la integridad de la información extraída. La recuperación de archivos es un componente crucial del análisis forense y también encuentra aplicación en dispositivos defectuosos. Para este propósito, existen diversas técnicas y herramientas para recuperar datos, que van desde simples restauraciones de la papelera de reciclaje hasta operaciones de recuperación de disco más complejas. En este trabajo, investigará varios sistemas de archivos, pero nos centramos en el sistema FAT por ser el más utilizado en "pendrives". Para este sistema, estudiamos las ventajas y limitaciones de varias técnicas, utilizando herramientas de código abierto. El resultado de este trabajo será una herramienta con las siguientes funcionalidades:

- Acceso a los clústeres en crudo, además de otras técnicas como el 'File carving'.
- Recuperación de información del sistema, como la tabla de particiones MBR, y otros ficheros como el 'Bash history'.

Esta herramienta presenta un peligro en unidades de almacenamiento desechadas conteniendo información confidencial, por ello se estudiará el borrado de los bytes de una unidad, conocido como 'File wiping', en conjunción con está herramienta

---

File recovering in filesystems

Abstract

keywords: Forensic Analysis, FAT

Computer forensics is a computer science discipline that deals with law enforcement issues in the public domain. This analysis, which is performed on computers and other computing devices, is for legal purposes and it is essential to ensure the integrity of the information extracted. File recovery is a crucial component of forensic analysis and also finds application in defective devices. For this purpose, various techniques and tools are available to recover data, ranging from simple recycle garbage can restores to more complex disk recovery operations. In this bachelor thesis, we investigate several file systems, but we focus on the FAT system because it is the most commonly used on pen drives. For this filesystem, we study the advantages and limitations of various techniques, using open source tools. The result of this work will be a tool with the following functionalities:

- Access to raw clusters, in addition to other techniques such as 'File carving'.
- Retrieval of system information, such as the MBR partition table, and other files such as 'Bash history'.

This tool presents a danger on discarded storage drives containing sensitive information, so the deletion of bytes from a drive, known as 'File wiping', will be studied in conjunction with this tool.

Índice

1 Introducción

En el mundo actual, nuestra dependencia de la información es más profunda que nunca. Desde la comunicación y el entretenimiento hasta el trabajo y la educación, prácticamente todos los aspectos de nuestra vida cotidiana están influenciados por la información que consumimos, generamos y compartimos a través de sistemas informáticos. Por ejemplo, utilizamos sistemas informáticos para almacenar y acceder a datos personales, documentos importantes, datos bancario, correos electrónicos y cualquier medio multimedia digital.

Esta dependencia hacia la información nos lleva a confiar en la disponibilidad y la integridad de los datos que almacenamos en sistemas informáticos. Sin embargo, esta confianza puede verse comprometida por una serie de factores, desde casos más simples como errores humanos o fallos técnicos hasta peligros mayores como ataques cibernéticos o desastres naturales. En estos momentos críticos es cuando la recuperación de información cobra una importancia vital.

Es por esto que la recuperación de información en sistemas informáticos es crucial para garantizar la disponibilidad, integridad y seguridad de los datos en un mundo cada vez más dependiente de la información digital. Cuando los datos se corrompen, se eliminan o se pierde el acceso a ellos por cualquier motivo, la capacidad de recuperarlos de manera efectiva se vuelve esencial. Este proceso de recuperación implica la aplicación de técnicas y herramientas especializadas para restaurar los datos perdidos o dañados, asegurando así la continuidad de las operaciones y la preservación de al información.

En entornos empresariales, la pérdida de datos puede tener consecuencias financieras significativas, afectar la continuidad del negocio y comprometer la seguridad de la información sensible. Por lo tanto, la capacidad de recuperar información de manera rápida y eficiente se convierte en una prioridad para las organizaciones, que implementan estrategias y soluciones de recuperación de datos para mitigar los riesgos asociados con la pérdida de datos.

Los sistemas de archivos son las estructuras utilizadas para organizar y almacenar información en dispositivos de almacenamiento y, por tanto, son la pieza fundamental en la gestión de la información en sistemas informáticos. Estos sistemas definen cómo se almacenan, nombran, acceden y manipulan los archivos en un dispositivo de almacenamiento.

Para poder gestionar de la mejor manera estos sistemas de archivos existen distintos mecanismos de organización, en este trabajo nos centraremos en el uso de las tablas FAT.

Las tablas FAT (File Allocation Table) son un componente importante en la gestión de datos en sistemas de archivos como FAT12, FAT16 y FAT32, utilizados en una variedad de dispositivos de almacenamiento. Estas tablas proporcionan un mapa que indica la ubicación de cada archivo en el dispositivo de almacenamiento, facilitando así el proceso de recuperación de datos al permitir a los sistemas operativos localizar y restaurar los archivos perdidos o dañados. Serían por poner un ejemplo práctico el ”índice”del sistema donde se almacena la información.

Herramientas

Para llevar a cabo este programa todas las pruebas se han realizado en un una maquina virtual Oracle VM Virtualbox de sistema operativo Linux.

Al tratarse de un estudio sobre la recuperación de información en sistemas de ficheros con File Allocation Tables de 16 Bits (FAT16), se han empleado distintas técnicas y métodos relacionados con la recuperación forense de la información, los más relevantes para este proyecto serían los siguientes:

1. **Fatcat:**
   Fatcat es una herramienta utilizada para explorar, analizar y modificar sistemas de archivos FAT. Es útil para recuperar datos, examinar la estructura del sistema de archivos.

2. **Foremost:**
   Foremost es una herramienta de recuperación de datos diseñada para extraer archivos de una imagen de disco basada en encabezados, pies de página y estructuras de datos internas. Se suele utilizar en ciencia forense digital para recuperar archivos borrados.

3. **Fdisk:**
   Fdisk es una herramienta utilizada para manipular la tabla de particiones de un disco duro. Permite crear, eliminar, modificar y listar particiones en un disco, siendo una herramienta fundamental en la administración de discos en sistemas Linux.

4. **grep y su uso en la búsqueda de strings:**
   grep (Global Regular Expression Print) es una herramienta utilizada para buscar patrones en texto. Se puede usar para buscar strings específicos o patrones definidos por expresiones regulares en archivos.

Para ayudar al lector, hemos incluido una serie de términos técnicos que usaremos a lo largo de este trabajo.

1. **Codificaciones ASCII y UTF-8:**

   a) ASCII (American Standard Code for Information Interchange):
   Es un estándar de codificación de caracteres que utiliza 7 bits para representar 128 caracteres diferentes, incluyendo letras, números y símbolos comunes. Aunque al estar limitada el inglés suele estar acompañada de UTF-8.

   b) UTF-8 (Unicode Transformation Format - 8 bits):
   Es una codificación de caracteres que puede representar cualquier carácter del conjunto Unicode utilizando entre 1 y 4 bytes. Es compatible con ASCII (los primeros 128 caracteres son los mismos) y es ampliamente utilizada debido a su eficiencia y capacidad para representar caracteres de todos los idiomas.

2. **Serialización:**
   La serialización es el proceso de convertir una estructura de datos o un objeto en un formato que pueda ser fácilmente almacenado o transmitido y luego reconstruido. La serialización es esencial para guardar el estado de un objeto en un archivo, transmitir datos entre sistemas o enviar datos a través de redes.

3. **Números mágicos para los tipos de fichero:**
   Un número mágico es una secuencia específica de bytes al inicio de un archivo que indica su tipo. Por ejemplo, los archivos JPEG comienzan con los bytes FF D8 FF, y los archivos PDF comienzan con %PDF. Estos números permiten a los programas identificar rápidamente el tipo de archivo sin depender del tamaño del archivo.

4. **Sectores (físicos y lógicos) y bloques:**

   a) **Sectores físicos:**
   Son las unidades más pequeñas de almacenamiento en un disco físico, generalmente de 512 bytes o 4096 bytes (4KB). Se refieren a cómo los datos están organizados físicamente en la superficie del disco.

   b) **Sectores lógicos:**
   Son utilizados por el sistema operativo para interactuar con el disco. Pueden coincidir en tamaño con los sectores físicos o ser agrupaciones de varios sectores físicos.

   c) **Bloques:**
   En sistemas de archivos, un bloque es una unidad de almacenamiento que agrupa varios sectores lógicos. El tamaño de bloque comúnmente es de 4KB y es la unidad mínima de lectura y escritura en muchos sistemas de archivos.

Objetivos y alcance del Trabajo

Para la realización de este proyecto he optado por utilizar un modelo de cascada, ya que al ser un trabajo individual, me permitía dictaminar bien los pasos a seguir y realizarlos en orden para un correcto desarrollo del Proyecto.

En primer lugar, determiné los requisitos que necesitaría mi proyecto mediante una entrevista con mi director. Los requisitos del proyecto fueron fijados a los siguientes:

Los problemas a resolver incluyen la recuperación de ficheros y proporcionar scripts para el borrado seguro de datos, asegurando que los scripts de recuperación no sean efectivos.

Las características del sistema incluyen acceso a datos en crudo sin procesamiento, recuperación de archivos basándose en su contenido y estructura sin la necesidad de información de la tabla de archivos, capacidad para reparar tablas MBR y otros ficheros, funcionalidad para borrar de forma segura la información, ser de código abierto, compatibilidad con codificaciones comunes y reconocimiento de cabeceras de diferentes tipos de fichero.

Las entradas del sistema son dispositivos de almacenamiento que utilizan el sistema de archivos FAT 16, los archivos perdidos o borrados a recuperar y la información a borrar en caso de utilizar la funcionalidad de borrado seguro.

Las salidas incluyen archivos recuperados, información del sistema recuperada como la tabla de particiones MBR y otros ficheros, y dispositivos de almacenamiento seguros tras el borrado seguro de información.

El sistema de archivos que se utilizará es el sistema FAT 16. El sistema deberá acceder al dispositivo de almacenamiento para leer y analizar su contenido, procesar información del sistema como la tabla de particiones MBR y permitir el borrado seguro de información confidencial.

La función deseada es un diseño sencillo sin interfaz gráfica, utilizando comandos por terminal y adaptable a otros sistemas de archivo. Las interfaces establecidas se limitarán a la línea de comandos en Bash. Esta herramienta está pensada para ser utilizada en el ámbito académico como por ejemplo para ser usado por alumnos de ciberseguridad con conocimientos mínimos de informática.

Las restricciones que afectan al proyecto incluyen entre otras que el diseño debe ser exclusivo para el sistema de archivos FAT 16, la dependencia de la integridad de los datos y el estado físico del dispositivo para la recuperación de archivos, y la necesidad de asegurar la efectividad del borrado seguro contra técnicas avanzadas de recuperación.

Para llevar a cabo el proyecto, nos hemos ayudado en gran medida con los contenidos teóricos del libro Lin y Lagerstrom-Fife [2018], además de utilizar los libros Ling [2013] y los artículos Mechtly et al. [2019] y Chen y mei Liu [2010] a modo de refuerzo. Para la estructura del estudio, se ha decidido dividir el trabajo en 3 capítulos distintos.

El capítulo 2, se ha centrado en técnicas de recuperación estándar para cualquier sistema de ficheros, como búsqueda de cadenas de texto, identificación de cabeceras de distintos tipos de archivo, recuperación de información borrada y modificación y sobrescritura en la información de la memoria. Esto se ha hecho de esta manera para tener un conocimiento y bases aplicables a todo tipo de sistemas de ficheros, para luego poder aplicarlos en el sistema en específico en el que se centra el trabajo.

En el capítulo 3 que abarca el grueso del proyecto, se dedica específicamente al funcionamiento, almacenamiento, borrado y recuperación de información de los sistemas de ficheros que funcionan con File Allocation Tables de 16 Bits (FAT-16). En esta parte la explicación se centra en la estructura y funcionamiento de manera teórica para después realizar la pruebas pertinentes en un sistema de ficheros de Sistema Operativo Linux, además de la programación de scripts de código bash que permita una automatización de la recuperación de información en ficheros fragmentados y el borrado de los mismos.

Por último, en el capítulo 4 se estudia el análisis de cómo los sistemas de archivos VFAT organizan sus distintos ficheros mediante el renombramiento interno de los distintos archivos para la optimización del espacio de memoria en las File Allocation Table, haciendo uso de entradas Long File Name (LFN) y entradas estándar 8.3.

2 Recuperación de ficheros: Bases y Herramientas

Sistemas de ficheros

Un dispositivo de almacenamiento es un conjunto de componentes que proporcionan a los sistemas informáticos un espacio dividido en Bytes que permiten realizar acciones de lectura y escritura de datos.

.

Este espacio por si solo cuenta con un poco de organización interna, sin embargo, teniendo en cuenta la cantidad de datos que se suelen guardar en un sistema informático se opta por emplear un elemento especializado en almacenar y manejar mejor la información, llamado sistema de ficheros.

Estos sistemas disponen de una base de datos en la que se registran todo los datos relevantes a los ficheros que se van agregando al sistema, como sus nombres, su extensión o su localización en el espacio de almacenamiento, entre otros.

Para visualizar de mejor manera el como se guardan cada archivo dependiendo de su tipo y su contenido, se crearán 3 archivos de 3 tipos diferentes: Una imagen en formato PNG con nombre accidente.png, un documento PDF con nombre documento.pdf y un simple archivo de texto llamado notas.txt. Después se añadirá un disco de imagen virtual de 2,5G de tamaño. Los dispositivos en Linux son todos tratados como archivos y, en este caso, la imagen virtual recibe el nombre de sdb y está en el directorio dev.

```
pablo@localhost:/$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0    7:0    0     4K  1 loop /snap/bare/5
loop1    7:1    0  74.2M  1 loop /snap/core22/1380
loop2    7:2    0 269.6M  1 loop /snap/firefox/4173
loop3    7:3    0  10.7M  1 loop /snap/firmware-updater/127
loop4    7:4    0 505.1M  1 loop /snap/gnome-42-2204/176
loop5    7:5    0  91.7M  1 loop /snap/gtk-common-themes/1535
loop6    7:6    0  10.3M  1 loop /snap/snap-store/1124
loop7    7:7    0  38.7M  1 loop /snap/snapd/21465
loop8    7:8    0   476K  1 loop /snap/snapd-desktop-integration/157
sda      8:0    0    25G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0  10.5G  0 part
└─sda3   8:3    0  14.5G  0 part /var/snap/firefox/common/host-hunspell
                                  /
sr0     11:0    1  1024M  0 rom
pablo@localhost:/$
```

Ilustración 1: Estado inicial del disco duro del Sistema

Una vez se tiene el disco, se crea una partición de 1G, y es en esta partición sdb1 donde se hará el análisis de los archivos.

```
pablo@localhost:/$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0    7:0    0     4K  1 loop /snap/bare/5
loop1    7:1    0  74.2M  1 loop /snap/core22/1380
loop2    7:2    0 269.6M  1 loop /snap/firefox/4173
loop3    7:3    0  10.7M  1 loop /snap/firmware-updater/127
loop4    7:4    0 505.1M  1 loop /snap/gnome-42-2204/176
loop5    7:5    0  91.7M  1 loop /snap/gtk-common-themes/1535
loop6    7:6    0  38.7M  1 loop /snap/snapd/21465
loop7    7:7    0  10.3M  1 loop /snap/snap-store/1124
loop8    7:8    0   476K  1 loop /snap/snapd-desktop-integration/157
sda      8:0    0    25G  0 disk
├─sda1   8:1    0     1M  0 part
├─sda2   8:2    0  10.5G  0 part
└─sda3   8:3    0  14.5G  0 part /var/snap/firefox/common/host-hunspell
                                 /
sdb      8:16   0   2.5G  0 disk
└─sdb1   8:17   0     1G  0 part
```

Ilustración 2: Adición del disco sdb y de la partición sdb1 del mismo.

Para que una partición pueda ser utilizada, primero se formatea para que partición FAT 16 y luego se "monta" en un directorio para trabajar en ella. En este caso, se ha creado el directorio parte1 dentro del directorio media, que es el usado normalmente por el sistema operativo. Cabe destacar que, pese a tratarse de un sistema FAT16, todas las pruebas y comandos utilizados en este capítulo se pueden adaptar a cualquier sistema de ficheros en un sistema Linux y, por tanto, no se centran específicamente en la estructura o funcionamiento del sistema en sí.

```
pablo@localhost:~$ sudo mkfs.fat -F 16 /dev/sdb1
```

Ilustración 3: Creación del sistema de ficheros de tipo FAT 16 con nombre /dev/sdb1

```
pablo@localhost:~$ sudo mount /dev/sdb1 /media/parte1/
```

Ilustración 4: Montado del sistema /dev/sdb1 en el directorio /media/parte1

```
pablo@localhost:/media/parte1$ ls -l /media/parte1/
total 4416
-rwxr-xr-x 1 root root   40110 May  6 02:33 accidente.png
-rwxr-xr-x 1 root root 4445978 May  6 02:33 documento.pdf
-rwxr-xr-x 1 root root    1102 May  6 02:32 notas.txt
pablo@localhost:/media/parte1$
```

Ilustración 5: Contenido del sistema recién creado, con el tamaño de cada archivo expresado en Bytes.

Como se puede observar hay una gran diferencia entre el tamaño que ocupan en memoria, siendo el PDF mucho más pesado que el menor de ellos como es el archivo de texto.

Con el programa 'fatcat', se obtiene un listado de los ficheros guardados, que recoge también el número de clúster de bytes donde comienza el contenido de cada fichero. Además, gracias a esto, se puede conocer la dirección de cada clúster.

```
pablo@localhost:/media/parte1$ sudo fatcat /dev/sdb1 -l /
Listing path /
Directory cluster: 0
f 6/5/2024 16:40:00  notas.txt (NOTAS.TXT)                    c=3 s=1102 (1.07617K)
f 6/5/2024 16:40:08  documento.pdf (DOCUME~1.PDF)             c=4 s=4445978 (4.24002M)
f 6/5/2024 16:40:22  accidente.png (ACCIDE~1.PNG)             c=276 s=40110 (39.1699K)
```

Ilustración 6: Número de sector el que se ubica el clúster donde comienza el contenido de los archivos del sistema /dev/sdb1

```
pablo@localhost:/media/parte1$ sudo fatcat /dev/sdb1 -@ 3
Cluster 3 address:
311296 (000000000004c000)
Next cluster:
FAT1: 4294967295 (ffffffff)
FAT2: 4294967295 (ffffffff)
Chain size: 1 (16384 / 16K)
Chain is contiguous
pablo@localhost:/media/parte1$ sudo fatcat /dev/sdb1 -@ 4
Cluster 4 address:
327680 (0000000000050000)
Next cluster:
FAT1: 5 (00000005)
FAT2: 5 (00000005)
Chain size: 272 (4456448 / 4.25M)
Chain is contiguous
pablo@localhost:/media/parte1$ sudo fatcat /dev/sdb1 -@ 276
Cluster 276 address:
4784128 (0000000000490000)
Next cluster:
FAT1: 277 (00000115)
FAT2: 277 (00000115)
Chain size: 3 (49152 / 48K)
Chain is contiguous
```

Ilustración 7: Información de cada uno de los archivos (notas.txt, documento.pdf y hexdump.png respectivamente). Dirección de memoria y número de clúster donde comienzan a almacenarse en memoria, el tamaño que ocupan en total medido en número de clústers ocupados, Bytes y KBytes/MegaBytes y la continuidad o no de los datos.

Comparando el archivo de texto con los otros dos, se puede ver como tienen un tamaño muy superior. Esto se debe a que en memoria la información se guarda en forma de una cadena de Bytes continua y los archivos .pdf o las imágenes .png están formados por muchos más elementos que una cadena de texto. Por lo tanto, a la hora de traducirlos para poder guardar la información, el espacio necesario es mucho mayor. Sobretodo apreciable en el documento.pdf, que ocupa 272 veces más que el archivo notas.txt.

Recuperación de ficheros borrados

Lo que resulta más chocante a primera vista sobre los comandos que se emplean habitualmente para borrar archivos es que estos no borran nada en realidad. Se entiende por borrar eliminar la información que ocupa el espacio en memoria y no dejar rastro de él, dando como resultado espacio libre en memoria. Sin embargo, hacer esto sería demasiado costoso en recursos, ya que llevaría demasiado tiempo, por lo que se opta por algo más sencillo. El sistema de ficheros basa toda su estructura en su tabla de particiones, por lo que si algún archivo no apareciera en esta, este sería inaccesible. Y es en esto mismo

en lo que se basan los comandos como rm, que “borran” archivos. En vez de eliminar completamente toda la información relativa al fichero, modifican su entrada en la tabla de particiones eliminando su nombre y dirección e indicando el espacio como libre, provocando que el sistema no tenga un registro de dónde se encuentra la información del fichero y, por tanto, se “olvide” de este. Este proceso se llama en inglés ‘ùnlink”.
De esta manera, la información del fichero “eliminado” no se pierde pero se vuelve inaccesible, ya que el sistema, al no reconocer que hay información relevante en esa parte de la memoria, considera ese espacio como disponible. Es así cómo, los datos del fichero “borrado” permanecen en memoria hasta que esta información es sobrescrita por otros datos. Esta técnica se conoce como “lazy deletion” y se utiliza por su eficiencia: en vez de gastar tiempo en eliminar los datos, simplemente se elimina del índice la entrada donde están registrados.

Por está razón se puede recuperar ficheros, ya que si no se ha almacenado nueva información en el lugar que ocupa el contenido de un fichero ”borrado”, ese contenido sigue siendo legible. Tras haber eliminado los 3 archivos haciendo uso del comando rm, ya no aparecen en el contenido del directorio. Sin embargo, se puede comprobar que, accediendo a la dirección en el espacio de almacenamiento de los tres ficheros, los datos siguen ahí:

```
pablo@localhost:/media/parte1$ sudo hexdump -C /dev/sdb1 -s 311296 -n 16384
0004c000  43 6f 6e 20 64 69 65 7a  20 63 61 c3 b1 6f 6e 65  |Con diez ca..one|
0004c010  73 20 70 6f 72 20 62 61  6e 64 61 0a 56 69 65 6e  |s por banda.Vien|
0004c020  74 6f 20 65 6e 20 70 6f  70 61 20 61 20 74 6f 64  |to en popa a tod|
0004c030  61 20 76 65 6c 61 0a 4e  6f 20 63 6f 72 74 61 20  |a vela.No corta |
0004c040  65 6c 20 6d 61 72 20 73  69 20 6e 6f 20 76 75 65  |el mar si no vue|
0004c050  6c 61 0a 55 6e 20 76 65  6c 65 72 6f 20 62 65 72  |la.Un velero ber|
0004c060  67 61 6e 74 c3 ad 6e 0a  42 61 6a 65 6c 20 70 69  |gant..n.Bajel pi|
0004c070  72 61 74 61 20 71 75 65  20 6c 6c 61 6d 61 6e 0a  |rata que llaman.|
0004c080  50 6f 72 20 73 75 20 62  72 61 76 75 72 61 20 65  |Por su bravura e|
0004c090  6c 20 74 65 6d 69 64 6f  0a 45 6e 20 74 6f 64 6f  |l temido.En todo|
pablo@localhost:~$ sudo hexdump -C /dev/sdb1 -s 327680 -n 200
00050000  25 50 44 46 2d 31 2e 35  0a 25 e4 f0 ed f8 0a 31  |%PDF-1.5.%.....1|
00050010  37 20 30 20 6f 62 6a 0a  3c 3c 2f 54 79 70 65 2f  |7 0 obj.<</Type/|
00050020  50 61 74 74 65 72 6e 2f  50 61 74 74 65 72 6e 54  |Pattern/PatternT|
00050030  79 70 65 20 31 2f 50 61  69 6e 74 54 79 70 65 20  |ype 1/PaintType |
00050040  32 2f 54 69 6c 69 6e 67  54 79 70 65 20 32 2f 42  |2/TilingType 2/B|
00050050  42 6f 78 5b 30 20 30 20  31 30 30 20 34 5d 2f 58  |Box[0 0 100 4]/X|
00050060  53 74 65 70 20 31 30 30  2f 59 53 74 65 70 0a 34  |Step 100/YStep.4|
00050070  2f 52 65 73 6f 75 72 63  65 73 3c 3c 3e 3e 2f 4d  |/Resources<<>>/M|
00050080  61 74 72 69 78 5b 2e 35  20 2d 2e 38 36 36 30 32  |atrix[.5 -.86602|
00050090  35 20 2e 38 36 36 30 32  35 20 2e 35 20 30 20 30  |5 .866025 .5 0 0|
pablo@localhost:/media/parte1$ sudo hexdump -C /dev/sdb1 -s 4784120 -n 49152
0048fff8  00 00 00 00 00 00 00 00  89 50 4e 47 0d 0a 1a 0a  |.........PNG....|
00490008  00 00 00 0d 49 48 44 52  00 00 03 84 00 00 02 33  |....IHDR.......3|
00490018  08 03 00 00 00 6f 28 27  70 00 00 00 a8 50 4c 54  |.....o('p....PLT|
00490028  45 e6 e6 e6 48 00 6c 32  00 05 43 03 0a ff f7 50  |E...H.l2..C....P|
00490038  0d d1 d7 ff ff 55 ff ff  ff ff 26 1c 44 04 74 ff  |.....U....&.D.t.|
00490048  14 12 2f 00 63 39 00 6b  fa fe f3 11 c2 cb f0 f2  |../.c9.k........|
00490058  ed 16 af be ff fa 43 1b  9b b1 ff 2f 20 21 87 a3  |......C..../ !..|
00490068  3e 1c 2c 38 29 84 30 4a  92 59 1f 82 35 3b 4a 28  |>.,8).0J.Y..5;J(|
00490078  6a a0 26 75 83 2e 57 66  6e 3d 92 ff 4b 34 c9 ba  |j.&u..Wfn=..K4..|
00490088  d2 b8 a3 c4 d9 d0 de 97  75 aa a9 8d b7 72 0c 15  |........u....r..|
```

Ilustración 8: Muestra el contenido de la partición /dev/sdb1, comenzando desde el byte 311296 para notas.txt, el byte 327680 para documento.pdf y el byte 4784120 para el accidente.png, mostrando estos datos en formato hexadecimal y ASCII

En caso de desconocer la dirección de los datos cuya entrada se ha eliminado de la tabla, la estrategia habitual consiste en recorrer el espacio de memoria hasta encontrar algún fragmento de interés. El ejemplo más conocido es hacer búsqueda por cadena de texto o de expresiones regulares y en caso de encontrarla, obtener su dirección.

```
pablo@localhost:/$ cd media/parte1/
pablo@localhost:/media/parte1$ ls
pablo@localhost:/media/parte1$ sudo grep -abo "a toda vela" /dev/sdb1
311339:a toda vela
pablo@localhost:/media/parte1$
```

Ilustración 9: Búsqueda del string “a toda vela” en todo el sistema, que coincide con una parte de los restos en memoria tras el ”borrado”de notas.txt

Existen multitud de herramientas para recuperar ficheros basadas en la misma idea: efectúan un

barrido del dispositivo de almacenamiento en búsqueda de cabeceras o “footers” de los formatos más habituales (JPEG, AVI, PDF, etc.), en lo que se conoce como “file carving”.

```
pablo@localhost:/media/parte1$ sudo foremost -t pdf -i /dev/sdb1
Processing: /dev/sdb1
|***********|
pablo@localhost:/media/parte1$ ls
output
pablo@localhost:/media/parte1$ cd output/
pablo@localhost:/media/parte1/output$ ls
audit.txt  pdf
pablo@localhost:/media/parte1/output$ cd pdf/
pablo@localhost:/media/parte1/output/pdf$ ls
00000640.pdf  00009504.pdf
pablo@localhost:/media/parte1/output/pdf$
```

```
pablo@localhost:/media/parte1$ sudo foremost -t jpg -t png -i /dev/sdb1
Processing: /dev/sdb1
|***********|
pablo@localhost:/media/parte1$ ls
output
pablo@localhost:/media/parte1$ cd output/
pablo@localhost:/media/parte1/output$ ls
audit.txt  jpg  png
pablo@localhost:/media/parte1/output$ cd png/
pablo@localhost:/media/parte1/output/png$ ls
00009344.png
pablo@localhost:/media/parte1/output/png$ cd ..
pablo@localhost:/media/parte1/output$ cd jpg/
pablo@localhost:/media/parte1/output/jpg$ ls
00000843.jpg  00009707.jpg
pablo@localhost:/media/parte1/output/jpg$
```

Está es la técnica que los programas como 'foremost' utilizan. Se limitan a buscar contenidos concretos, sin ocuparse de los metadatos en la tabla del sistema de ficheros (nombre, tamaño, localización) que puedan permanecer tras el borrado.

```
pablo@localhost:/media/parte1$ sudo fatcat /dev/sdb1 -l /
Listing path /
Directory cluster: 0
d 6/5/2024 18:58:42  output/ (OUTPUT)                                    c=1087
pablo@localhost:/media/parte1$ sudo fatcat /dev/sdb1 -l / -d
[sudo] password for pablo:
Listing path /
Directory cluster: 0
d 6/5/2024 18:58:42  output/ (OUTPUT)                                    c=1087
f 6/5/2024 16:40:08  documento.pdf (OCUME~1.PDF)                         c=4 s=4445978
 (4.24002M) d
f 6/5/2024 16:40:22  accidente.png (CCIDE~1.PNG)                         c=276 s=40110
 (39.1699K) d
```

«Tachado» de ficheros

Para evitar que los datos almacenados sean legibles, podemos «taparlos» sobrescribiendo en la misma posición de memoria: desde anular todos los valores poniéndolos a ceros o directamente TACHANDO la información.

```
pablo@localhost:~$ sudo chmod a+w /dev/sdb1
pablo@localhost:~$ sudo hexdump -C /dev/sdb1 -s 327680 -n 100
00050000  25 50 44 46 2d 31 2e 35  0a 25 e4 f0 ed f8 0a 31  |%PDF-1.5.%.....1|
00050010  37 20 30 20 6f 62 6a 0a  3c 3c 2f 54 79 70 65 2f  |7 0 obj.<</Type/|
00050020  50 61 74 74 65 72 6e 2f  50 61 74 74 65 72 6e 54  |Pattern/PatternT|
00050030  79 70 65 20 31 2f 50 61  69 6e 74 54 79 70 65 20  |ype 1/PaintType |
00050040  32 2f 54 69 6c 69 6e 67  54 79 70 65 20 32 2f 42  |2/TilingType 2/B|
00050050  42 6f 78 5b 30 20 30 20  31 30 30 20 34 5d 2f 58  |Box[0 0 100 4]/X|
00050060  53 74 65 70                                       |Step|
00050064
pablo@localhost:~$ sudo dd if=/dev/zero of=/dev/sdb1 oflag=seek_bytes seek=32768
0 count=100 > iflag=count_bytes conv=notrunc
100+0 records in
100+0 records out
51200 bytes (51 kB, 50 KiB) copied, 0.00438859 s, 11.7 MB/s
pablo@localhost:~$ sudo hexdump -C /dev/sdb1 -s 327680 -n 100
00050000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00050060  00 00 00 00                                       |....|
00050064
```

Ilustración 10: Antes y después de la sustitución de la información en memoria mediante la sobrescritura de ceros para el borrado completo de la información

```
pablo@localhost:~$ yes -- "--TACHADO--" | sudo dd of=/dev/sdb1 oflag=seek_bytes
seek=327680 count=100 > iflag=count_bytes conv=notrunc
100+0 records in
100+0 records out
51200 bytes (51 kB, 50 KiB) copied, 0.00756217 s, 6.8 MB/s
pablo@localhost:~$ sudo hexdump -C /dev/sdb1 -s 327680 -n 100
00050000  2d 2d 54 41 43 48 41 44  4f 2d 2d 0a 2d 2d 54 41  |--TACHADO--.--TA|
00050010  43 48 41 44 4f 2d 2d 0a  2d 2d 54 41 43 48 41 44  |CHADO--.--TACHAD|
00050020  4f 2d 2d 0a 2d 2d 54 41  43 48 41 44 4f 2d 2d 0a  |O--.--TACHADO--.|
00050030  2d 2d 54 41 43 48 41 44  4f 2d 2d 0a 2d 2d 54 41  |--TACHADO--.--TA|
00050040  43 48 41 44 4f 2d 2d 0a  2d 2d 54 41 43 48 41 44  |CHADO--.--TACHAD|
00050050  4f 2d 2d 0a 2d 2d 54 41  43 48 41 44 4f 2d 2d 0a  |O--.--TACHADO--.|
00050060  2d 2d 54 41                                       |--TA|
00050064
```

Ilustración 11: Sobrescritura más visual con el texto '–TACHADO–'

Sin embargo, aunque la información del fichero está tapada por los ceros, el nombre del archivo sigue siendo legible.

```
pablo@localhost:~$ sudo fatcat /dev/sdb1 -l / -d
Listing path /
Directory cluster: 0
d 6/5/2024 18:58:42  output/ (OUTPUT)                                  c=1087
f 6/5/2024 16:40:08  documento.pdf (OCUME~1.PDF)                       c=4 s=4445978
 (4.24002M) d
f 6/5/2024 16:40:22  accidente.png (CCIDE~1.PNG)                       c=276 s=40110
 (39.1699K) d
```

Para prevenir ataques más sofisticados de recuperación de archivos», lo más normal es sobrescribir la información con bits aleatorios para no revelar información sobre el tamaño del fichero, además de hacerlo varias veces para borrar rastros electromagnéticos Ling [2013].

```
pablo@localhost:~$ for N in $(seq 3)
> do
> sudo dd if=/dev/urandom of=/dev/sdb1 oflag=seek_bytes seek=327680 count=100
> iflag=count_bytes conv=notrunc
> done
[sudo] password for pablo:
100+0 records in
100+0 records out
51200 bytes (51 kB, 50 KiB) copied, 0.00344642 s, 14.9 MB/s
100+0 records in
100+0 records out
51200 bytes (51 kB, 50 KiB) copied, 0.00303284 s, 16.9 MB/s
100+0 records in
100+0 records out
51200 bytes (51 kB, 50 KiB) copied, 0.00268525 s, 19.1 MB/s
pablo@localhost:~$ sudo hexdump -C /dev/sdb1 -s 327680 -n 100
00050000  01 a5 63 e6 99 88 36 9b  b0 bd f1 6a 05 d4 1b 34  |..c...6....j...4|
00050010  50 48 cb 64 97 7f 7a 33  ec ac f6 e4 36 b6 2a 1e  |PH.d..z3....6.*.|
00050020  95 e0 00 9e 56 f9 cf f1  c1 48 98 09 0c ea 48 8a  |....V....H....H.|
00050030  a3 7a dd 54 a9 93 48 21  bc 15 75 3d 4d a2 9b 10  |.z.T..H!..u=M...|
00050040  65 3d b9 68 bf 7d 77 81  ed 1f 7b 28 eb 60 a2 fe  |e=.h.}w...{(.`..|
00050050  8e 58 71 d2 ed 5f 3b 28  9e 9c 1e e8 cf f2 b8 f1  |.Xq.._;(........|
00050060  d9 2a 61 e8                                       |.*a.|
```

3 Técnicas específicas para FAT

Preparación del entorno

Para la realización de las pruebas se comienza con un dispositivo en blanco. Para ello simplemente se añadirá un nuevo disco duro al sistema que se llamará automáticamente lsb, en este caso se trata´de un disco VDI (Virtual Disk Image) de 2,5 Gibibytes de tamaño y con un tamaño de memoria fijo (Pre-Allocate Full Size). Una vez hecho esto, el disco estará recién creado, por lo tanto, ya se encuentra vacío para comenzar con nuestro análisis de las características del disco:

```
pablo@localhost:~$ sudo /sbin/fdisk -l /dev/sdb
Disk /dev/sdb: 2.5 GiB, 2685634560 bytes, 5245380 sectors
Disk model: VBOX HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x9a6fb4cf

Device     Boot Start     End Sectors Size Id Type
/dev/sdb1        2048 2099199 2097152   1G 83 Linux
```

Ilustración 12: Características de disco sdb

Haciendo esto se obtienen los datos más básicos de el dispositivo, en este caso se puede ver cómo el dispositivo cuenta con un total de 5245380 sectores, de 512 Bytes cada uno. Por último para calcular el tamaño total del Dispositivo solo es necesaria una simple multiplicación:

$$245380 \times 512 = 2685634560\text{Bytes} \quad \approx 2{,}5 \text{ Gibiabytes}$$

Sabiendo que el tamaño seleccionado para el disco es de 2,5 GiB, se confirma que el disco se ha creado de manera correcta.

A modo de segunda comprobación se obtiene la misma información de la estructura del disco utilizando los siguientes comandos de blockdev:

```
pablo@localhost:~$ sudo blockdev --getsz /dev/sdb
5245380
pablo@localhost:~$ sudo blockdev --getpbsz /dev/sdb
512
pablo@localhost:~$ sudo blockdev --getsize64 /dev/sdb
2685634560
```

Ilustración 13: valores de el número de Sectores, tamaño de sector y tamaño en Bytes del disco usando el comando blockdev.

Aunque las mismas pruebas se realizarán en está partición, se ha creado una partición de tipo FAT16 con un tamaño de 10 MiB. Esto ha se ha hecho así para poder realizar pruebas de forma más ágil.

Al tratarse de FAT16, las líneas de memoria se organizan en 16 Bytes. Con esta instrucción, se está utilizando para la tabla de particiones el formato tradicional *Master Boot Record* (*disk label* DOS, opción por defecto para `sfdisk`), en lugar del más moderno *GUID Partition Table* (*disk label* GPT).

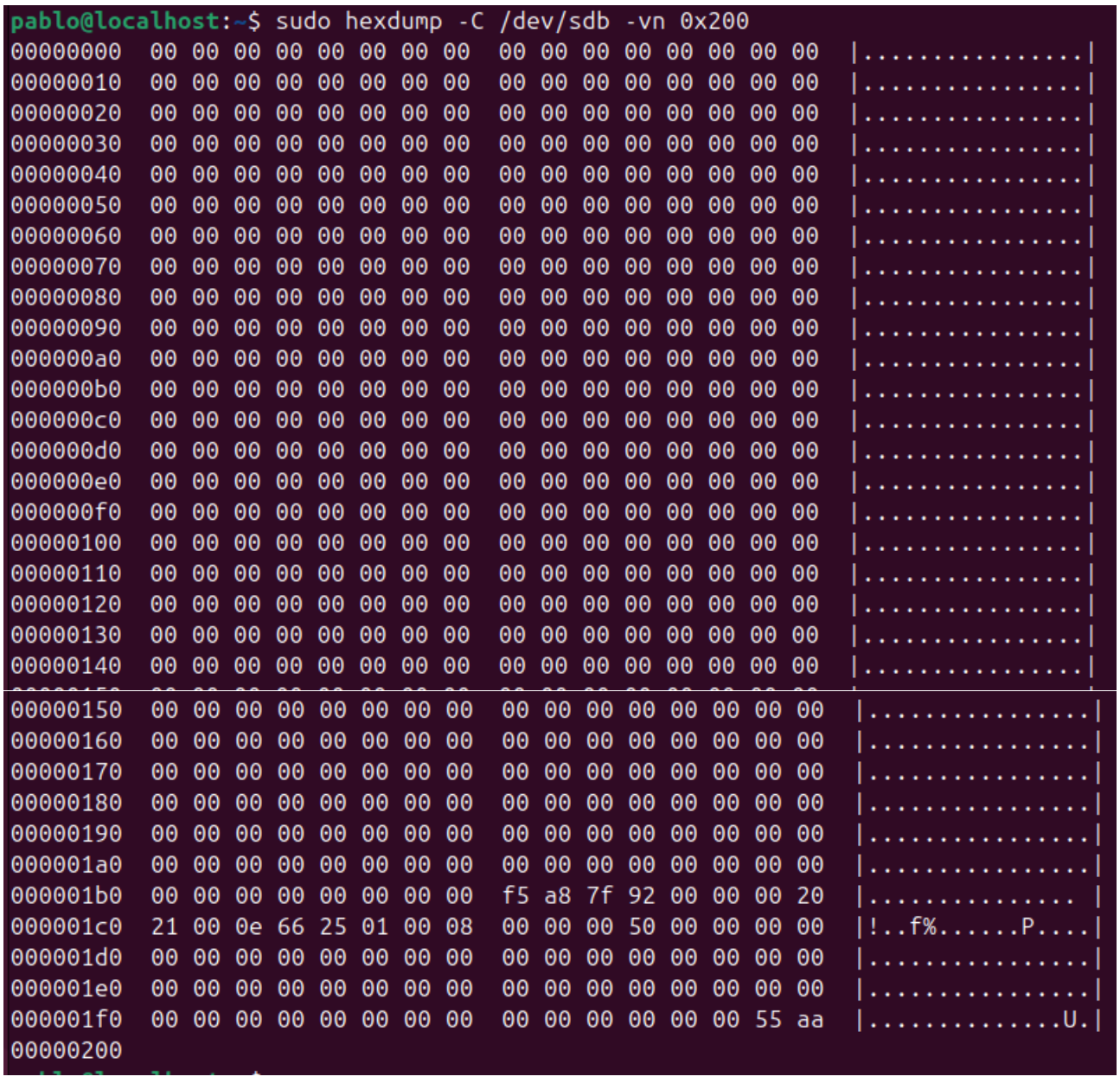


Ilustración 14: Datos que hay en la partición recién creada.

En el formato de Master Boot Record, la información relativa a cada partición se codifica en dieciséis bytes.

Organización de las particiones FAT16

El siguiente comando se utiliza para la generación del formato del dispositivo.

```
# /sbin/mkfs.fat /dev/sdb1
```

Está instrucción prepara un sistema de ficheros de tipo FAT en el dispositivo elegido, lo que supone utilizar algunos de sus 10 MiB para escribir la información necesaria.

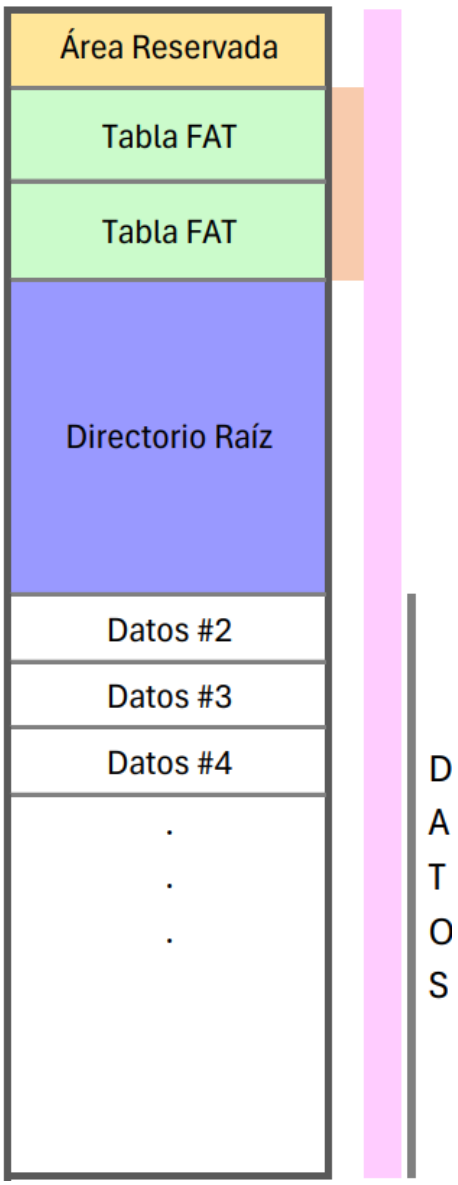
Los sistemas de archivos FAT16 tienen una estructura básica dividida en 4 secciones:

1. **Sector de Arranque (Boot Sector):**
   También conocido como Sector de Inicio o Sector de Arranque Maestro (Master Boot Record, MBR). Este sector se encuentra en el primer sector del disco y contiene información crucial para iniciar el sistema y para el sistema de archivos FAT16. Incluye el código de inicio, la tabla de particiones que indica cómo está dividido el disco y la estructura específica del sistema de archivos FAT16.

2. **Tabla de Asignación de Archivos (File Allocation Table, FAT):**
   La FAT es una estructura de datos crucial que registra la asignación de clústeres en el disco. En FAT16, la FAT es una matriz donde cada entrada de la matriz (de 16 bits) corresponde a un clúster en el disco. La FAT contiene información sobre qué clústeres están ocupados por archivos y cuáles están libres. A modo de prevención de errores, es común que haya dos tablas FAT en único sistema de archivos a modo de copia de seguridad.

3. **Directorio Raíz (Root Directory):**
   Es un directorio especial ubicado inmediatamente después de la FAT. Contiene entradas para los archivos y subdirectorios ubicados directamente en el directorio raíz del disco. Cada entrada en el directorio raíz contiene información como el nombre del archivo, atributos, tamaño y la ubicación del primer clúster del archivo.

4. **Área de Datos:**
   Es la parte principal del disco que contiene los archivos y directorios. Los archivos se dividen en clústeres (generalmente de tamaño fijo), y la FAT se utiliza para seguir qué clústeres pertenecen a qué archivos.



El sistema de ficheros FAT organiza el espacio de la partición en clústeres de sectores, convirtiéndose el clúster en la unidad mínima de almacenamiento para cualquier uso. El Partition Boot Sector de un sistema de archivos FAT16 es una estructura crítica ubicada en el primer sector de cada partición. Contiene información sobre el formato del sistema de archivos y como se codifican varios parámetros que determinan cómo se reparten esos clústeres entre las distintas áreas de la partición. Se estructura de la siguiente manera:

Desplazamiento	Tamaño (Bytes)	Descripción
0x000 - 0x002	3	Instrucción de salto
0x003 - 0x00A	8	Nombre OEM
0x00B - 0x00C	2	Bytes por sector
0x00D	1	Sectores por clúster
0x00E - 0x00F	2	Sectores reservados
0x010	1	Número de tablas FAT
0x011 - 0x012	2	Entradas de directorio raíz
0x013 - 0x014	2	Número total de sectores (si < 65536)
0x015	1	Descriptor de medio
0x016 - 0x017	2	Sectores por FAT
0x018 - 0x019	2	Sectores por pista
0x01A - 0x01B	2	Cabezas
0x01C - 0x01F	4	Sectores ocultos
0x020 - 0x023	4	Número total de sectores (si >= 65536)
0x024	1	Unidad BIOS
0x025	1	Sin usar (Reservado para Windows NT)
0x026	1	Firma extendida
0x027 - 0x02A	4	Número de serie del volumen
0x02B - 0x035	11	Etiqueta del volumen
0x036 - 0x03D	8	Sistema de ficheros
0x03E - 0x1FD	480	Código de arranque
0x1FE - 0x1FF	2	Firma de Arranque

Ilustración 15: Explicación de la estructura del boot section de cada partición, destacando los bloques relevantes para este trabajo.

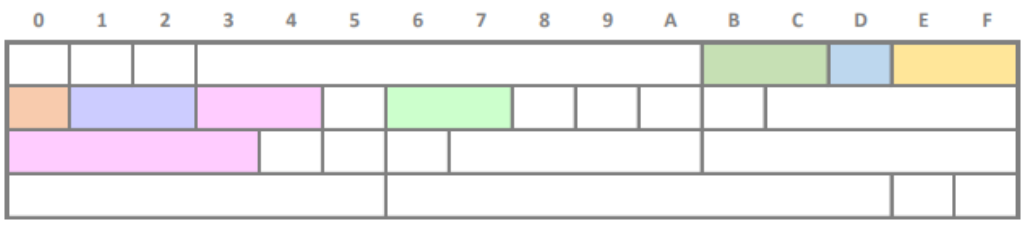


Ilustración 16: Representación física de la distribución que ocupa el boot section en memoria.

A continuación de las tablas FAT, un espacio de tamaño fijo para el directorio raíz. Se trata de una cantidad de bloques de 32 bytes, cada uno de los cuales registra los datos de un fichero o de otro directorio, (nombre, su tamaño, timestamps, algunos tipo de archivo o la localización del contenido en el sistema).

El contenido de un fichero (o de un directorio) se almacena en una sucesión de clústeres. Para «encontrar» el contenido de ese fichero, la entrada que lo declara en su directorio registra el (índice del) primero de esos clústeres (lo que antes se ha llamado localización).

La tabla FAT sirve para proseguir esa sucesión, indicando, para cada clúster, cuál le sigue. Como hemos indicado arriba, los clústeres de datos se numeran desde #2 en adelante: #2, #3, #4, etc. Cada uno se corresponde con una entrada en la tabla FAT, dos bytes en este caso al tratarse de FAT16.

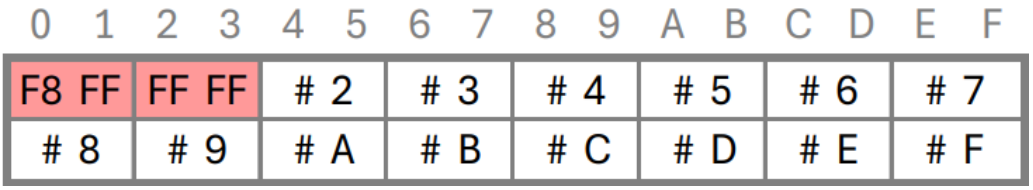


Ilustración 17: Caption

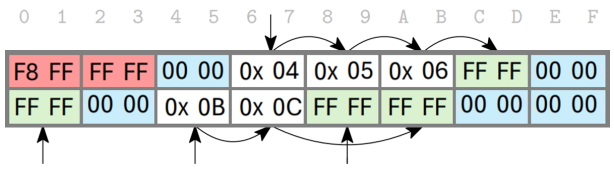
Los valores principales que puede tomar la casilla asociada a un clúster son:

1. El índice del clúster siguiente;

2. FF FF: si se trata de un clúster «terminal»

3. 00 00: si el clúster está disponible.

Haciendo esto, indicamos los clústeres especiales (que no cuentan con información del fichero), poniendo todos los bits de la entrada a 1 para indicar el final de un fichero. Y los clústeres sin ninguna información asociada se marca con ceros en su entrada de la tabla. De esta manera, la entrada de la tabla asociada a un clúster registra el clúster que le sucede. Por lo que, si adjuntamos a cada clúster de datos su entrada asociada en la tabla FAT, podemos interpretar la sucesión de clústeres donde se almacena datos como una lista enlazada.



#3 → #4 → #5 → #6 → END
#8 → END
#10 → #11 → #13 → END
#12 → END

Para poder realizar las pruebas correctamente, a continuación haciendo uso del comando od, se obtienen los valores relativos a los valores que necesitaremos.

```
pablo@localhost:~$ sudo od -A n -t u2 /dev/sdb1 -j 0x0B -N 2
   512
pablo@localhost:~$ sudo od -A n -t u1 /dev/sdb1 -j 0x0D -N 1
   4
pablo@localhost:~$ sudo od -A n -t u2 /dev/sdb1 -j 0x0E -N 2
     4
pablo@localhost:~$ sudo od -A n -t u1 /dev/sdb1 -j 0x10 -N 1
   2
pablo@localhost:~$ sudo od -A n -t u2 /dev/sdb1 -j 0x11 -N 2
   512
pablo@localhost:~$ sudo od -A n -t u2 /dev/sdb1 -j 0x13 -N 2
 20475
pablo@localhost:~$ sudo od -A n -t u2 /dev/sdb1 -j 0x16 -N 2
    20
pablo@localhost:~$
```

Ilustración 18: Valor de número de Bytes en cada sector, sectores en cada clúster, sectores RESERVADOS, número de TABLAS FAT, máximo de entradas en directorio raíz, número total de sectores y sectores en cada Tabla FAT.

Forzando la fragmentación de un fichero

Simulación de Disco Completo

Para poder realizar las pruebas de recuperación de información fragmentada, es necesario primero contar con un dispositivo con información. En este caso vamos a aprovechar la partición /dev/sdb1 de 10 MiB que hemos usado en el apartado anterior.

Calcularemos ahora la cantidad de clústeres de datos vacíos.

En el caso de nuestro ejemplo, la partición (de 10 MiB) cuenta con 20475 sectores, agrupados en clústeres indivisibles. Por lo que tenemos 20480 / 4 = 5118 clústeres en total.

El área reservada está constituida por 4 sectores, o lo que es lo mismo, un único clúster.

Cada tabla FAT tiene asignados 0x14 (20) sectores , es decir que ocupa 20 sectores/4 sectores por clúster = 5 clústeres por tabla. AL tener 2 Tablas FAT, tenemos 10 clústeres dedicados a tablas FAT.

El directorio raíz admite un máximo de 0x2000 (512) entradas, considerando apuntes «tradicionales» de 32 bytes (que permiten codificar nombres de fichero con ocho caracteres y una extensión de tres).

La extensión del área designada para (el contenido de) el directorio raíz es, por lo tanto, el tamaño necesario para que el directorio raíz esté completo (512 entradas * 32 bytes por entrada) dividido entre los bytes que caben en un clúster (Bytes que caben en un sector * número de sectores que forman un clúster):

$$\frac{512 \cdot 32}{512 \cdot 4} = \frac{32}{4} = 8 \text{ clústeres.}$$

En consecuencia, el espacio que resta para el contenido de los ficheros y de los directorios distintos del raíz es de

$$5118 - 1 - 10 - 8 = 5099 \text{ clústeres.}$$

A continuación para comprobar que los cálculos teóricos se aproximan al caso real, he programado un script bash que se encarga de hacer el cálculo con los datos obtenidos:

```bash
#!/bin/bash

# Dispositivo de la partición FAT16 (modificar según sea necesario)
DISPOSITIVO="/dev/sdb1"

# Obtener los valores necesarios del sector de arranque
BYTES_POR_SECTOR=$(od -A n -t u2 $DISPOSITIVO -j 0x0B -N 2)
SECTORES_POR_CLUSTER=$(od -A n -t u1 $DISPOSITIVO -j 0x0D -N 1)
SECTORES_RESERVADOS=$(od -A n -t u2 $DISPOSITIVO -j 0x0E -N 2)
NUMERO_DE_FATS=$(od -A n -t u1 $DISPOSITIVO -j 0x10 -N 1)
ENTRADAS_DEL_DIRECTORIO=$(od -A n -t u2 $DISPOSITIVO -j 0x11 -N 2)
TOTAL_SECTORES=$(od -A n -t u2 $DISPOSITIVO -j 0x13 -N 2)
SECTORES_POR_FAT=$(od -A n -t u2 $DISPOSITIVO -j 0x16 -N 2)

# Mostrar los valores obtenidos para depuración
echo "BYTES_POR_SECTOR: $BYTES_POR_SECTOR"
echo "SECTORES_POR_CLUSTER: $SECTORES_POR_CLUSTER"
echo "SECTORES_RESERVADOS: $SECTORES_RESERVADOS"
echo "NUMERO_DE_FATS: $NUMERO_DE_FATS"
echo "ENTRADAS_DEL_DIRECTORIO: $ENTRADAS_DEL_DIRECTORIO"
echo "TOTAL_SECTORES: $TOTAL_SECTORES"
echo "SECTORES_POR_FAT: $SECTORES_POR_FAT"

# Calcular otros valores según los datos obtenidos
SECTORES_POR_FAT_TOTAL=$((SECTORES_POR_FAT * NUMERO_DE_FATS))
TAMANIO_DIRECTORIO_RAIZ_BYTES=$((ENTRADAS_DEL_DIRECTORIO * 32)) # Suponiendo que cada entrada es de 32 bytes

# Mostrar los valores calculados para depuración
echo "Sectores por tabla FAT: $SECTORES_POR_FAT_TOTAL"
echo "Tamaño del directorio raíz en bytes: $TAMANIO_DIRECTORIO_RAIZ_BYTES"

# Calcular clústeres disponibles para datos
if [[ $BYTES_POR_SECTOR -ne 0 && $SECTORES_POR_CLUSTER -ne 0 && $SECTORES_RESERVADOS -ne 0 ]]; then
  BYTES_POR_CLUSTER=$((BYTES_POR_SECTOR * SECTORES_POR_CLUSTER))
  CLUSTERS_RESERVADOS=$((SECTORES_RESERVADOS / SECTORES_POR_CLUSTER + SECTORES_POR_FAT_TOTAL / SECTORES_POR_CLUSTER))
  CLUSTERS_DIRECTORIO_RAIZ=$(( (TAMANIO_DIRECTORIO_RAIZ_BYTES) / BYTES_POR_CLUSTER )) # División redondeada hacia arriba
  TOTAL_CLUSTERS=$((TOTAL_SECTORES / SECTORES_POR_CLUSTER))
  CLUSTERS_DATOS_DISPONIBLES=$((TOTAL_CLUSTERS - CLUSTERS_RESERVADOS - CLUSTERS_DIRECTORIO_RAIZ))

  # Mostrar los resultados finales
  echo "Total de clústeres: $TOTAL_CLUSTERS"
  echo "Clústeres reservados: $CLUSTERS_RESERVADOS"
  echo "Clústeres del directorio raíz: $CLUSTERS_DIRECTORIO_RAIZ"
  echo "Clústeres de datos disponibles: $CLUSTERS_DATOS_DISPONIBLES"
else
  echo "Error: Se detectaron valores incorrectos. Verifique el dispositivo y el script."
fi
```

Listing 3.1: Clústeres disponibles para inforamción.

Caso de Uso	Descripción
Obtener valores del sector de arranque	Lee y extrae varios valores cruciales del sector de arranque de una partición FAT16.
Mostrar valores obtenidos	Imprime en la consola los valores obtenidos para depuración y verificación.
Calcular tamaño de las tablas FAT	Calcula el tamaño total en sectores de las tablas FAT sumando el tamaño de cada FAT multiplicado por el número de FATs.
Calcular tamaño del directorio raíz	Calcula el tamaño en bytes del directorio raíz en función del número de entradas y el tamaño de cada entrada.
Calcular clústeres disponibles para datos	Realiza varias operaciones para determinar el número de clústeres disponibles para almacenar datos.
Mostrar resultados finales	Imprime en la consola los resultados finales de los cálculos, incluyendo clústeres totales, reservados, del directorio raíz y disponibles para datos.
Manejar errores en la obtención de valores	Verifica que los valores críticos no sean cero y muestra un mensaje de error si se detectan valores incorrectos.

Ilustración 19: Casos de uso del script.

Ahora ejecutamos el código para comprobar si nuestros c cálculos son correctos:

```
pablo@localhost:~/Desktop$ chmod +x calcula_clusters_libres.sh
pablo@localhost:~/Desktop$ sudo ./calcula_clusters_libres.sh
BYTES_POR_SECTOR:    512
SECTORES_POR_CLUSTER:    4
SECTORES_RESERVADOS:      4
NUMERO_DE_FATS:    2
ENTRADAS_DEL_DIRECTORIO:    512
TOTAL_SECTORES:  20475
SECTORES_POR_FAT:     20
Sectores por tabla FAT: 40
Tamaño del directorio raíz en bytes: 16384
Total de clústeres: 5118
Clústeres reservados: 11
Clústeres del directorio raíz: 8
Clústeres de datos disponibles: 5099
pablo@localhost:~/Desktop$
```

Ilustración 20: Resultado del script que calcula los clústeres disponibles.

Como podemos comprobar al ejecutar el código podemos comprobar que efectivamente los cálculos son correctos.

A continuación, se copian en la partición con ficheros para ocupar todo el espacio y poder ver como funciona la desfragmentación.

En un primer momento, decidí utilizar 50 copias de un fichero que ocupe exactamente 100 clústeres. Dejando el resto de espacio a ser ocupado por un fichero más pequeño, ocupando este 99 clústeres. Sin embargo, tras hacer las pruebas, al tratarse de tamaños tan parecidos, aunque se veía una diferencia, no consideré que fuera lo suficientemente significativa como para explicar claramente el funcionamiento.

Es por esto que las pruebas al final fueron realizadas utilizando 51 fichero que ocupan 99 clústeres, y el espacio restante es ocupado por un fichero que ocupa 50.

Teniendo en cuenta esto, si cada clúster tiene 4 sectores y cada sector tiene 512 bytes, entonces el tamaño de cada clúster es de:

$$4 \text{ sectores/clúster} \times 512 \text{ bytes/}sector = 2048 \text{ bytes/clúster}$$

Con esto en mente, necesitamos calcular el tamaño de los archivos basándonos en este tamaño de clúster.

Para 99 clústeres:

$$99 \times 2048 = 202752 \quad \text{Bytes}$$

Para 50 clústeres:

$$50 \times 2048 = 102400 \quad \text{Bytes}$$

Lo primero es programar un script bash que genere los archivos necesarios para llenar el disco, en este caso será un programa que será utilizado por el otro script (encargado de llenar el disco) por lo que simplemente genera un archivo con contenido aleatorio proveniente de /dev/urandom al cual se le introduce dos parámetros: Su tamaño en Bytes y el nombre del archivo deseado.

```
#!/bin/bash

# Comprobar que se haya proporcionado el tamaño del archivo y el nombre del archivo
if [ $# -ne 2 ]; then
  echo "Uso: $0 <tamaño_en_bytes> <nombre_del_archivo>"
  exit 1
fi

TAMANO_BYTES=$1
NOMBRE_ARCHIVO=$2

# Crear un archivo del tamaño especificado con datos aleatorios
dd if=/dev/urandom of=$NOMBRE_ARCHIVO bs=$TAMANO_BYTES count=1

echo "Archivo $NOMBRE_ARCHIVO creado con tamaño $TAMANO_BYTES bytes con datos aleatorios"
```

Listing 3.2: Script que genera archivos con contenido aleatorio de tamaño fijo introducido en Bytes.

Caso de Uso	Descripción
Proporcionar parámetros necesarios	El usuario debe proporcionar dos parámetros: tamaño del archivo en bytes y nombre del archivo.
Validar la entrada del usuario	El script verifica que se hayan proporcionado exactamente dos argumentos, de lo contrario muestra un mensaje de uso y finaliza.
Asignar valores a variables	Asigna los valores proporcionados por el usuario a las variables TAMANO_BYTES y NOMBRE_ARCHIVO.
Crear archivo con datos aleatorios	Utiliza dd para crear un archivo con el nombre especificado y el tamaño en bytes, llenándolo con datos aleatorios del dispositivo /dev/urandom.
Confirmar la creación del archivo	Imprime un mensaje confirmando que el archivo se ha creado con el tamaño especificado y contiene datos aleatorios.

Ilustración 21: Casos de uso del script

Una vez obtenido el script encargado de generar los archivos, se programa el script responsable del llenado del disco.

En este script, el primer paso a seguir es la creación de dos ficheros, uno que ocupe 99 clústeres y otro que ocupa 50 clústeres.

Después, se crea el punto de montaje de la partición y se indica el disco que vamos a llenar, en este caso /dev/sdb1. Una vez hecho esto, y tras comprobar que el la partición se ha montado correctamente, solo queda insertar en el disco 51 copias del pseudotexto.txt (que ocupa 99 clústers) y por ultimo guardar el pseudotexto_2.txt que ocupa los 50 clústeres restantes.

```
#!/bin/bash

# Crear archivos de tamaño fijo con datos aleatorios
./tam_fijo.sh 202752 pseudotexto.txt
./tam_fijo.sh 102400 pseudotexto_2.txt

echo "Archivos pseudotexto.txt y pseudotexto_2.txt creados."

# Montar el dispositivo
MOUNT_POINT="/media/lab"
DEVICE="/dev/sdb1"

# Crear el punto de montaje si no existe
if [ ! -d "$MOUNT_POINT" ]; then
  sudo mkdir -p $MOUNT_POINT
fi

# Montar el dispositivo
sudo mount $DEVICE $MOUNT_POINT

# Comprobar si el montaje fue exitoso
if [ $? -ne 0 ]; then
  echo "Error: No se pudo montar el dispositivo $DEVICE en $MOUNT_POINT"
  exit 1
fi

echo "Dispositivo $DEVICE montado en $MOUNT_POINT"

# Copiar los archivos para llenar la partición
for N in $(seq -w 51); do
  sudo cp pseudotexto.txt $MOUNT_POINT/f_$N.txt
done

sudo cp pseudotexto_2.txt $MOUNT_POINT/f_52.txt

# Desmontar el dispositivo
sudo umount $MOUNT_POINT

echo "Relleno de la partición completado."

# Verificar el uso de espacio
sudo mount $DEVICE $MOUNT_POINT
sudo du -B 0x800 -c $MOUNT_POINT/*
sudo umount $MOUNT_POINT
```

Listing 3.3: Script que genera archivos aleatorios y los utiliza para llenar el espacio de detos de /dev/sdb1

Caso de Uso	Descripción
Crear archivos de tamaño fijo con datos aleatorios	Ejecuta otro script (tam_fijo.sh) para crear dos archivos (pseudotexto.txt y pseudotexto_2.txt) de tamaños específicos con datos aleatorios.
Crear punto de montaje	Verifica si el directorio de punto de montaje (/media/lab) existe; si no, lo crea.
Montar dispositivo	Monta un dispositivo especificado (/dev/sdb1) en el punto de montaje (/media/lab).
Verificar el éxito del montaje	Comprueba si el montaje del dispositivo fue exitoso, y si no, muestra un mensaje de error y termina el script.
Copiar archivos al dispositivo montado	Copia repetidamente los archivos generados al dispositivo montado para llenar la partición con múltiples copias.
Desmontar dispositivo	Desmonta el dispositivo después de copiar los archivos.
Verificar el uso de espacio	Vuelve a montar el dispositivo, verifica el uso de espacio en bloques de 2048 bytes, y desmonta de nuevo

Ilustración 22: Casos de uso del script.

Se puede realizar una comprobación de que el script ha funcionado, ya que imprime por pantalla todos los archivos creados con éxito.

```
pablo@localhost:~/Desktop$ chmod +x tam_fijo.sh
pablo@localhost:~/Desktop$ chmod +x llena_disco.sh
pablo@localhost:~/Desktop$ sudo ./llena_disco.sh
1+0 records in
1+0 records out
202752 bytes (203 kB, 198 KiB) copied, 0.00414865 s, 48.9 MB/s
Archivo pseudotexto.txt creado con tamaño 202752 bytes con datos aleatorios
1+0 records in
1+0 records out
102400 bytes (102 kB, 100 KiB) copied, 0.00584652 s, 17.5 MB/s
Archivo pseudotexto_2.txt creado con tamaño 102400 bytes con datos aleatorios
Archivos pseudotexto.txt y pseudotexto_2.txt creados.
Dispositivo /dev/sdb1 montado en /media/lab
Relleno de la partición completado.
```

Ilustración 23: Salida por pantalla que muestra el exito a la hora de rellenar el disco.

Ahora que el disco completamente lleno, vamos a comprobar como funciona al fragmentación de un fichero en el sistema.
Para ello, se liberará espacio en el disco eliminando dos de los archivos que acabamos de crear asegurándose de que no sean consecutivos. De esta manera, se obtendrá y almacenará un archivo que ocupe un tamaño mayor de 99 clústeres, provocando que se tenga que fragmentar.

Primero, liberaremos espacio en el disco haciendo uso de este script:

```
#!/bin/bash

# Configuración
DISPOSITIVO="/dev/sdb1"
MOUNT_POINT="/media/lab"

# Montar la partición
if ! mountpoint -q "$MOUNT_POINT"; then
  if sudo mount $DISPOSITIVO $MOUNT_POINT; then
    echo "Dispositivo montado en $MOUNT_POINT"
  else
```

```
    echo "Error al montar el dispositivo $DISPOSITIVO en $MOUNT_POINT"
    exit 1
  fi
else
  echo "El dispositivo ya está montado en $MOUNT_POINT"
fi

# Eliminar archivos especificados si existen
if [ -e "$MOUNT_POINT/f_10.txt" ]; then
  rm "$MOUNT_POINT/f_10.txt"
else
  echo "Archivo $MOUNT_POINT/f_10.txt no existe"
fi

if [ -e "$MOUNT_POINT/f_20.txt" ]; then
  rm "$MOUNT_POINT/f_20.txt"
else
  echo "Archivo $MOUNT_POINT/f_20.txt no existe"
fi

# Mostrar el uso del disco solo para los archivos dentro del directorio $MOUNT_POINT
du -B 0x800 -c "$MOUNT_POINT"/* | grep "$MOUNT_POINT"

# Desmontar la partición
if sudo umount $DISPOSITIVO; then
  echo "Dispositivo desmontado correctamente"
else
  echo "Error al desmontar el dispositivo $DISPOSITIVO"
  exit 1
fi
```

Listing 3.4: Script que elimina dos archivos no contiguos para generar espacio en el disco.

Caso de Uso	Descripción
Montar la partición si no está montada	Comprueba si el punto de montaje (/media/lab) ya está en uso, y si no, monta el dispositivo (/dev/sdb1) en ese punto.
Verificar y reportar el estado del montaje	Informa si el dispositivo se montó correctamente o si ya estaba montado, y maneja errores de montaje.
Eliminar archivos específicos si existen	Comprueba si los archivos (f_10.txt y f_20.txt) existen en el punto de montaje y los elimina; si no existen, informa al usuario.
Mostrar uso del disco para archivos en el directorio	Utiliza du para mostrar el uso del disco en bloques de 2048 bytes para los archivos en el directorio de montaje, filtrando la salida para incluir solo el directorio de montaje.
Desmontar la partición	Desmonta el dispositivo del punto de montaje y maneja posibles errores durante el proceso.

Ilustración 24: Casos de uso del script.

Este código elimina los ficheros, f_10.txt y el f_20.txt de /media/lab y muestra el contenido del directorio para poder verificar si se ha eliminado correctamente.

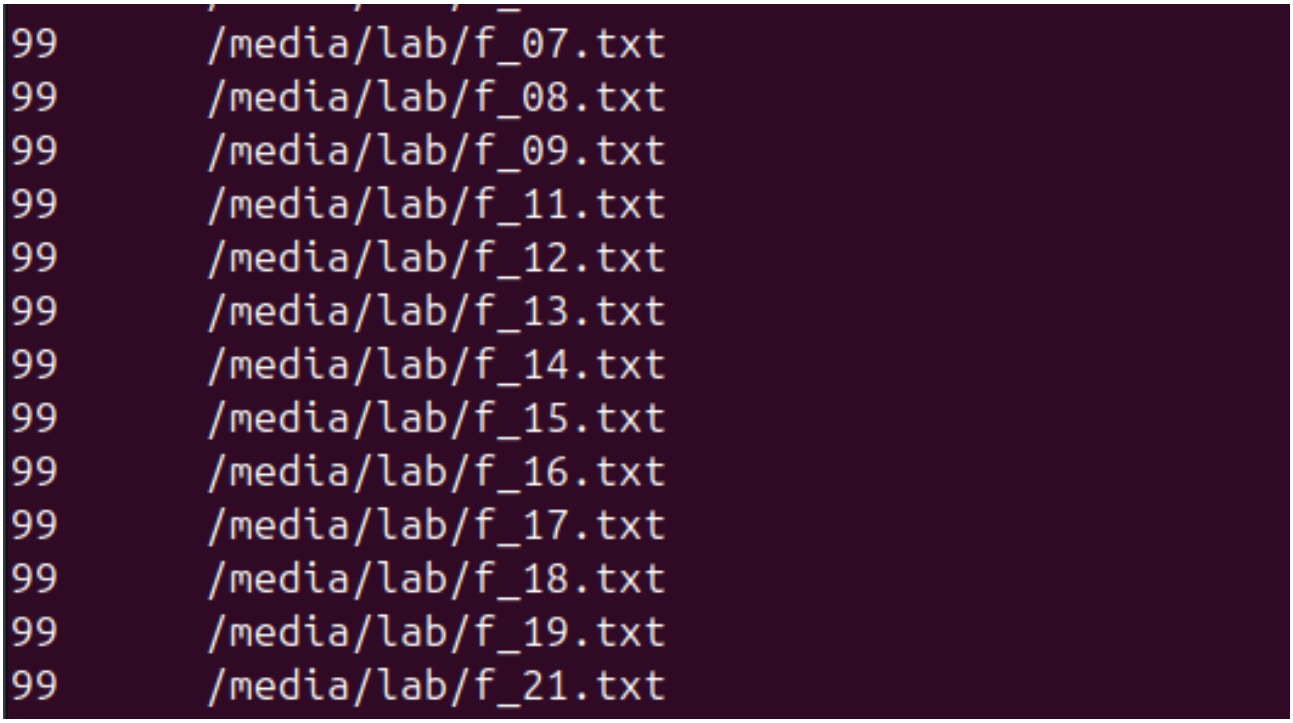


Ilustración 25: Comprobación de la eliminación de los ficheros f_10 y f_20 en el directorio /media/lab

De esta manera, se han ”borrado” (como se ha explicado en el capítulo 2, el contenido de los archivos sigue existiendo y está ocupando el mismo espacio, pero está marcado como no disponible y por tanto no se puede acceder a él) dos bloques de 99 clústeres cada uno.
Al haber creado todos los ficheros y el sistema en una partición vacía manualmente se da por hecho que los sectores que formaban los archivos están juntos entre sí (todos los sectores que contienen la información de f_10.txt están contiguos en la memoria, cosa que también ocurre con f_20.txt,) sin embargo, al haber eliminado dos archivos separados, está garantizado que si elegimos un archivo con tamaño superior a 99 clústeres, este se tendrá que fragmentar.

Para cumplir con el objetivo, obtendremos un fichero con algún formato (pdf, jpeg, png) con una extensión superior a 202752 Bytes y que no supere los 405504 Bytes para que haya espacio suficiente para guardarse. Si se cumplen estos requisitos al copiarlo a la partición, tendrá que fragmentarse inevitablemente en dos pedazos.

En este caso, se ha hecho uso de una imagen en formato pdf a la que llamaremos ‘Plano Península de la Magdalena.png’, que cuenta con 370,5 KB o lo que es lo mismo, 370500 Bytes.

Ahora tendremos que calcular cuántos clústeres ocupará nuestra imagen. En este caso al tener acceso al tamaño, se conoce que ocupa un total de 370500 Bytes, por lo que, siguiendo la misma lógica que hemos utilizado anteriormente, podemos averiguar cuántos clústeres ocupara de la siguiente manera:

$$\frac{370500 \text{ Bytes}}{2048 \text{ Bytes/Clúster}} = 180{,}908203125 \approx 181 \text{ clústeres}$$

Una vez hecho esto se copia el pdf a el directorio /media/lab en el que está montado /dev/sdb1 para comprobar en primer lugar si cabe, y si es así ver si se reescribe el espacio ”disponible”de manera correcta.

```
pablo@localhost:/media/lab$ sudo ls -la /media/lab
total 10184
drwxr-xr-x 2 root root  16384 Jan  1  1970  .
drwxr-xr-x 4 root root   4096 Jun 26 02:17  ..
-rwxr-xr-x 1 root root 370505 Jun 26 19:30 'Plano de la Península de La Magdalena.pdf'
-rwxr-xr-x 1 root root 202752 Jun 26 17:00  f_01.txt
-rwxr-xr-x 1 root root 202752 Jun 26 17:00  f_02.txt
-rwxr-xr-x 1 root root 202752 Jun 26 17:00  f_03.txt
-rwxr-xr-x 1 root root 202752 Jun 26 17:00  f_04.txt
-rwxr-xr-x 1 root root 202752 Jun 26 17:00  f_05.txt
-rwxr-xr-x 1 root root 202752 Jun 26 17:00  f_06.txt
-rwxr-xr-x 1 root root 202752 Jun 26 17:00  f_07.txt
```

Ilustración 26: Comprobación de que el pdf fue añadido con éxito al disco.

Para comprobar si los cálculos que se han realizado son correctos y si realmente el fichero se ha fragmentado, podemos hacer uso de fatcat en todo /media/lab para encontrar el clúster inicial del pdf y después fatcat en dicho clúster para ver si la cadena de caracteres se almacena de manera discontinua.

```
f 26/6/2024 17:30:16  Plano de la Península de La Magdalena.pdf (PLANOD~1.PDF) c=894 s=370505 (361.821K)
pablo@localhost:/media/lab$ sudo fatcat /dev/sdb1 -@ 894
Cluster 894 address:
1865728 (00000000001c7800)
Next cluster:
FAT1: 895 (0000037f)
FAT2: 895 (0000037f)
Chain size: 181 (370688 / 362K)
Chain is not contiguous
pablo@localhost:/media/lab$
```

Ilustración 27: Comprobación de que el pdf se ha añadido pero está dividido en varias partes ya que la cadena de caracteres aparece descontinua.

Como se puede observar el archivo pdf se ha añadido de manera correcta al sistema de ficheros y no solo eso, sino que también se ha conseguido forzar su fragmentación de manera correcta. Otro dato también ha destacar es que al tratarse del primer fragmento se indica el tamaño de cadena completo, el cual es, como bien habíamos calculado, 181 clústeres.

Con esto se descubre que el pdf comienza en el clúster número 894, por lo tanto si se conoce que ocupa más que los archivos iniciales, está garantizado que ocupará los primeros 99 clústeres que antes ocupaba el f_10.txt, por esto, se puede deducir que el primer fragmento de pdf ocupará desde el clúster número 894 hasta el $894 + 99 - 1 = 992$, eso se puede comprobar yendo hasta el clúster 992:

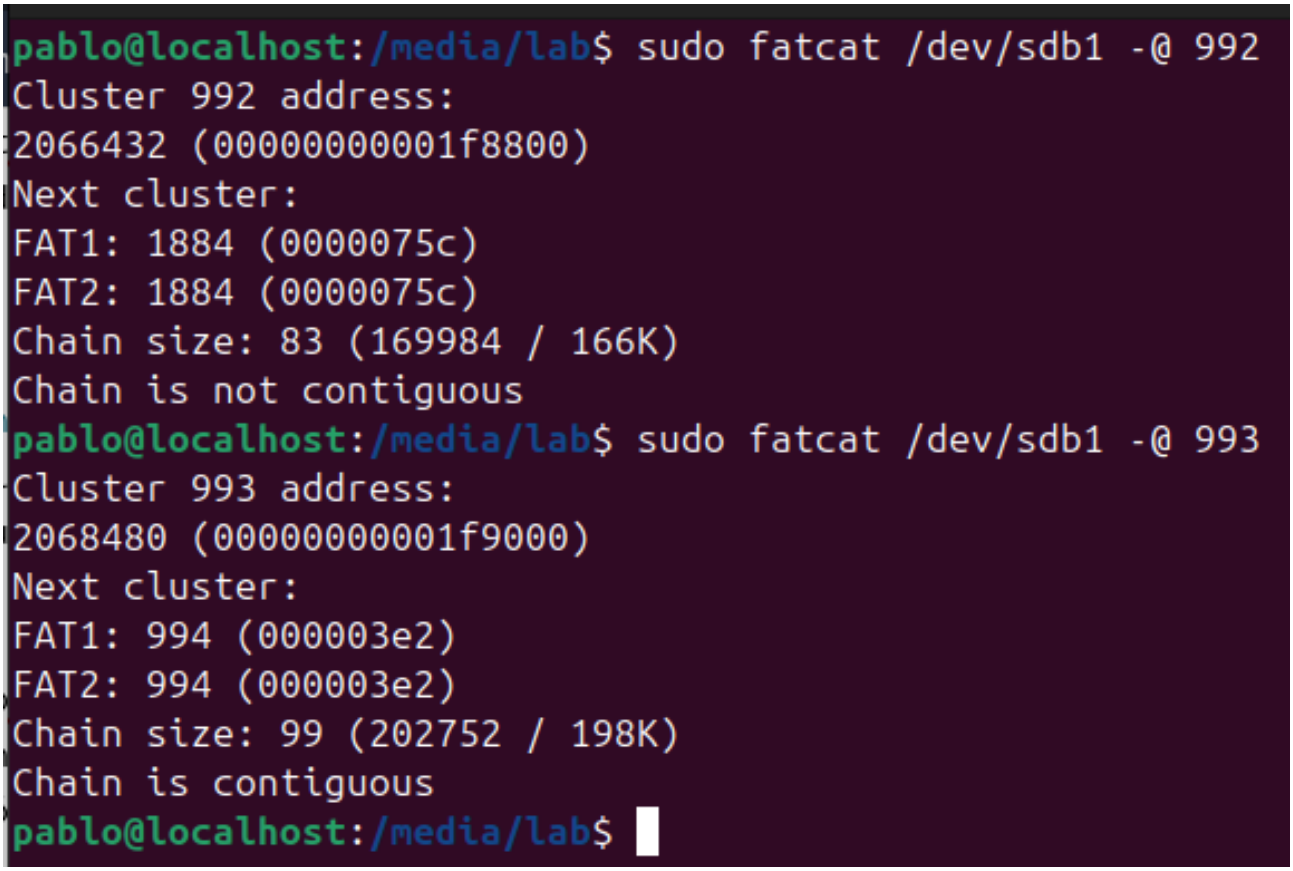


Ilustración 28: Muestra del último clúster del primer fragmento del pdf, viendo como el siguiente clúster pertenece a otro fichero que ya es contiguo.

Ahora habría que intentar encontrar el siguiente fragmento, en este caso se sabe que los fichero eliminados para dar espacio al pdf han sido f_10.txt y f_20.txt, eso nos da una pista de la separación que existe entre los dos fragmentos. Entre los dos .txt eliminados se encuentran los archivos f_11 al f_19.txt así que hay 9 archivos entre fragmento y fragmento, por lo que el numero de clústeres de por medio sería de $99 * 9 = 891$, esto lo que indica es que el siguiente fragmento se encuentra en el fragmento número $992 + 891 + 1 = 1884$ por lo que el siguiente fragmento comienza en el clúster número 1884. Lo único que restaría sería saber cuántos clústeres ocupa este segundo fragmento, cosa sencilla teniendo en cuenta que se saben conoce el número total de clústeres y el tamaño del primer fragmento, $181 - 99 = 82$ es el número de clústeres restantes.
Sabiendo esto, se puede calcular cuál es el último clúster: $1884 + 82 - 1 = 1965$. Ahora lo que resta es comprobarlo utilizando fatcat:

```
pablo@localhost:/media/lab$ sudo fatcat /dev/sdb1 -@ 1884
Cluster 1884 address:
3893248 (00000000003b6800)
Next cluster:
FAT1: 1885 (0000075d)
FAT2: 1885 (0000075d)
Chain size: 82 (167936 / 164K)
Chain is contiguous
pablo@localhost:/media/lab$ sudo fatcat /dev/sdb1 -@ 1965
Cluster 1965 address:
4059136 (00000000003df000)
Next cluster:
FAT1: 4294967295 (ffffffff)
FAT2: 4294967295 (ffffffff)
Chain size: 1 (2048 / 2K)
Chain is contiguous
pablo@localhost:/media/lab$
```

Ilustración 29: Primer y último clúster del segundo fragmento del pdf.

Como se puede observar, el segundo fragmento ocupa el rango de clústeres 1184-1965, esto se puede ver porque el clúster 1965 es el clúster que anuncia el final del archivo teniendo un valor de FFFF, además cabe destacar, que al tratarse del segundo y último fragmento del pdf, la cadena sí que es continua en este caso.

De esta manera se ha comprobado que el archivo pdf fragmentado ocupa 181 clúster distribuidos a lo largo de dos fragmentos, que ocupan los clústeres del 894 al 992 (1º fragmento) y del 1884 al 1965 (2º fragmento).

Borrado y Recuperación de Archivo

Utilizando el comando rm se borra el archivo pdf para intentar recuperarlo.

```
pablo@localhost:/media/lab$ sudo rm Plano\ de\ la\ Península\ de\ La\ Magdalena.
pdf
pablo@localhost:/media/lab$ ls
f_01.txt  f_08.txt  f_16.txt  f_24.txt  f_31.txt  f_38.txt  f_45.txt  f_52.txt
f_02.txt  f_09.txt  f_17.txt  f_25.txt  f_32.txt  f_39.txt  f_46.txt
f_03.txt  f_11.txt  f_18.txt  f_26.txt  f_33.txt  f_40.txt  f_47.txt
f_04.txt  f_12.txt  f_19.txt  f_27.txt  f_34.txt  f_41.txt  f_48.txt
f_05.txt  f_13.txt  f_21.txt  f_28.txt  f_35.txt  f_42.txt  f_49.txt
f_06.txt  f_14.txt  f_22.txt  f_29.txt  f_36.txt  f_43.txt  f_50.txt
f_07.txt  f_15.txt  f_23.txt  f_30.txt  f_37.txt  f_44.txt  f_51.txt
```

Ilustración 30: pdf eliminado del directorio

El primer intento consiste en el uso de foremost para recuperar parte de la información que pueda quedar en el disco.

```
pablo@localhost:~$ sudo foremost -i /dev/sdb1 -o /media/lab/recovery
Processing: /dev/sdb1
|*|
pablo@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  llena_disco.sh  snap
Documents  Music      Public    Videos     recovery
pablo@localhost:~$ ls /media/lab/recovery/
audit.txt
pablo@localhost:~$ cd /media/lab/
pablo@localhost:/media/lab$ ls
f_01.txt  f_08.txt  f_16.txt  f_24.txt  f_31.txt  f_38.txt  f_45.txt  f_52.txt
f_02.txt  f_09.txt  f_17.txt  f_25.txt  f_32.txt  f_39.txt  f_46.txt  recovery
f_03.txt  f_11.txt  f_18.txt  f_26.txt  f_33.txt  f_40.txt  f_47.txt
f_04.txt  f_12.txt  f_19.txt  f_27.txt  f_34.txt  f_41.txt  f_48.txt
f_05.txt  f_13.txt  f_21.txt  f_28.txt  f_35.txt  f_42.txt  f_49.txt
f_06.txt  f_14.txt  f_22.txt  f_29.txt  f_36.txt  f_43.txt  f_50.txt
f_07.txt  f_15.txt  f_23.txt  f_30.txt  f_37.txt  f_44.txt  f_51.txt
pablo@localhost:/media/lab$ cd recovery/
pablo@localhost:/media/lab/recovery$ ls
audit.txt
pablo@localhost:/media/lab/recovery$ ls -lh audit.txt
-rwxr-xr-x 1 root root 634 Jun 27 01:07 audit.txt
```

Ilustración 31: Uso de foremost en /dev/Sdb para ver si recupera el archivo o parte de este.

Lo que se obtiene al usar el comando, es un informe llamado audit.txt en el que se resumen los resultados de como ha recuperado la información.

```
00000140  2d 2d 2d 2d 2d 2d 2d 2d  2d 0a 46 69 6c 65 3a 20  |---------.File: |
00000150  2f 64 65 76 2f 73 64 62  31 0a 53 74 61 72 74 3a  |/dev/sdb1.Start:|
00000160  20 54 68 75 20 4a 75 6e  20 32 37 20 30 31 3a 30  | Thu Jun 27 01:0|
00000170  37 3a 33 34 20 32 30 32  34 0a 4c 65 6e 67 74 68  |7:34 2024.Length|
00000180  3a 20 31 30 20 4d 42 20  28 31 30 34 38 35 37 36  |: 10 MB (1048576|
00000190  30 20 62 79 74 65 73 29  0a 20 0a 4e 75 6d 09 20  |0 bytes). .Num. |
000001a0  4e 61 6d 65 20 28 62 73  3d 35 31 32 29 09 20 20  |Name (bs=512).  |
000001b0  20 20 20 53 69 7a 65 09  20 46 69 6c 65 20        |     Size. File |
000001c0  4f 66 66 73 65 74 09 20  43 6f 6d 6d 65 6e 74 20  |Offset. Comment |
000001d0  0a 0a 46 69 6e 69 73 68  3a 20 54 68 75 20 4a 75  |..Finish: Thu Ju|
000001e0  6e 20 32 37 20 30 31 3a  30 37 3a 33 34 20 32 30  |n 27 01:07:34 20|
000001f0  32 34 0a 0a 30 20 46 49  4c 45 53 20 45 58 54 52  |24..0 FILES EXTR|
00000200  41 43 54 45 44 0a 09 0a  2d 2d 2d 2d 2d 2d 2d 2d  |ACTED...--------|
00000210  2d 2d 2d 2d 2d 2d 2d 2d  2d 2d 2d 2d 2d 2d 2d 2d  |----------------|
*
```

Ilustración 32: Informe de foremost que muestra que no se encontró ningún archivo.

Al abrir y analizar este archivo lo que se obtiene son: malas noticias. Básicamente el comando no ha sido capaz de encontrar absolutamente nada sobre el archivo pdf que se ha eliminado recientemente.

Como segundo intento, se realizará la recuperación manual de la información haciendo uso de los datos que hemos conseguido antes analizando el pdf antes de su borrado.
Para la recuperación manual de los datos se hace uso de el comando hexdump y el comando dd para la recuperación de la información.

Primero cabe destacar antes de comenzar con la reconstrucción manual, que en caso de no haber analizado la distribución de los clústers antes de eliminar el archivo, la información del comando fatcat sería bastante menos eficaz. Esto se debe a que cuando se elimina un archivo en un sistema de archivos FAT, las entradas en la tabla FAT que apuntaban a los clústeres que formaban el archivo se marcan como libres (con valor 0), lo que hace que las cadenas que antes formaban parte del archivo ya no están necesariamente enlazadas de la misma manera. Así, cuando un archivo es eliminado, aunque los datos puedan seguir existiendo, el sistema de archivos ya no los enlaza como parte de un archivo, y las herramientas de análisis como fatcat mostrarán que las cadenas de clústers son discontinuas o cortas.

```
pablo@localhost:/media/lab/recovery$ sudo fatcat /dev/sdb1 -@ 894
Cluster 894 address:
1865728 (00000000001c7800)
Next cluster:
FAT1: 927 (0000039f)
FAT2: 927 (0000039f)
Chain size: 2 (4096 / 4K)
Chain is not contiguous
pablo@localhost:/media/lab/recovery$ sudo fatcat /dev/sdb1 -@ 895
Cluster 895 address:
1867776 (00000000001c8000)
Next cluster:
FAT1: 0 (00000000)
FAT2: 0 (00000000)
Chain size: 2 (4096 / 4K)
Chain is not contiguous
pablo@localhost:/media/lab/recovery$ sudo fatcat /dev/sdb1 -@ 992
Cluster 992 address:
2066432 (00000000001f8800)
Next cluster:
FAT1: 0 (00000000)
FAT2: 0 (00000000)
Chain size: 2 (4096 / 4K)
Chain is not contiguous
```

Ilustración 33: fatcat con valores a 0 para las tablas FAT.

Haciendo uso de Hexdump y dd se comienza a intentar la reconstrucción manual, para ello tenemos que tener en cuenta el tamaño en bytes de los clústeres 4 sectores/clúster $*$ 512 Bytes por sector $*$ OFFSET en este caso el offsett es el valor numérico del clúster, es decir, como ejemplo: Se ha determinado que el primer clúster del archivo pdf es el número 894, por lo tanto para hacer uso de estos comandos se le pondrá en la posición número $894 * 2028 = 1830912$ bytes.

Sabiendo esto se calculan los valores de posición de los clústeres del archivo pdf que se quieren recuperar: Primer Fragmento (894 - 992): Offset = 894 * 2048 = 1835008 bytes Offset = 992 * 2048 = 2031616 bytes Segundo Fragmento (1884 al 1965): Offset = 1884 * 2048 = 3858432 bytes Offset = 1965 * 2048 = 4024320 bytes

Tamaño de los Fragmentos: Tamaño 1 = (992 - 894 + 1) * 2048 = 200192 bytes Tamaño 2 = (1965 - 1884 + 1) * 2048 = 167936 bytes

Para la recuperación de la información se ha programado un script que recoge los dos segmentos que aun se mantienen en los clústeres aunque estén marcados como disponibles, antes de que se sobrescriba algo. Esto se consigue haciendo uso de el comando sudo dd y un directorio temporal donde meter los fragmentos recuperados, ya que solo existe espacio suficiente en /media/lab para que quepa o bien el pdf o bien los dos fragmentos.

```bash
#!/bin/bash

# Configuraciones
DISPOSITIVO="/dev/sdb1"
DESTINO="/media/lab/recovered_file.pdf"
TEMP_DIR=$(mktemp -d)

# Extracción de fragmentos
echo "Extrayendo primer fragmento (clusters 894 al 992)..."
dd if="$DISPOSITIVO" of="$TEMP_DIR/fragmento1.bin" bs=512 skip=$((894*4)) count=$((99*4))

echo "Extrayendo segundo fragmento (clusters 1884 al 1965)..."
dd if="$DISPOSITIVO" of="$TEMP_DIR/fragmento2.bin" bs=512 skip=$((1884*4))
    count=$((82*4))

# Verificación de la extracción
echo "Verificación de los fragmentos extraídos..."
ls -l $TEMP_DIR/fragmento1.bin
ls -l $TEMP_DIR/fragmento2.bin

# Unión de fragmentos
echo "Uniendo los fragmentos en $DESTINO..."
cat "$TEMP_DIR/fragmento1.bin" "$TEMP_DIR/fragmento2.bin" > "$DESTINO"

# Limpieza
echo "Limpiando directorio temporal: $TEMP_DIR"
rm -r "$TEMP_DIR"

echo "Proceso completado. Archivo recuperado en: $DESTINO"
```

Listing 3.5: Script que recupera el pdf eliminado

Caso de Uso	Descripción
Configurar variables	Define las variables DISPOSITIVO, DESTINO, y TEMP_DIR para especificar el dispositivo, destino del archivo recuperado y un directorio temporal.
Crear directorio temporal	Crea un directorio temporal para almacenar fragmentos extraídos usando mktemp.
Extraer primer fragmento de datos	Utiliza dd para extraer un fragmento de datos desde los clústeres 894 al 992 del dispositivo y lo guarda en un archivo temporal.
Extraer segundo fragmento de datos	Utiliza dd para extraer otro fragmento de datos desde los clústeres 1884 al 1965 del dispositivo y lo guarda en un archivo temporal.
Verificar la extracción de fragmentos	Lista los archivos extraídos en el directorio temporal para asegurar que se hayan creado correctamente.
Unir fragmentos de datos	Combina los fragmentos extraídos en un solo archivo y lo guarda en el destino especificado.
Limpiar el directorio temporal	Elimina el directorio temporal y todos los archivos en él para limpiar el entorno.
Informar la finalización del proceso	Muestra un mensaje indicando que el proceso de recuperación y unión de fragmentos ha sido completado, y el archivo recuperado se encuentra en el destino especificado.

Ilustración 34: Casos de uso del script.

```
pablo@localhost:~/Desktop$ chmod +x recupera_pdf.sh
pablo@localhost:~/Desktop$ sudo ./recupera_pdf.sh
Extrayendo primer fragmento (clusters 894 al 992)...
396+0 records in
396+0 records out
202752 bytes (203 kB, 198 KiB) copied, 0.00240129 s, 84.4 MB/s
Extrayendo segundo fragmento (clusters 1884 al 1965)...
328+0 records in
328+0 records out
167936 bytes (168 kB, 164 KiB) copied, 0.0019407 s, 86.5 MB/s
Verificación de los fragmentos extraídos...
-rw-r--r-- 1 root root 202752 Jun 27 04:22 /tmp/tmp.Pq2s2IxJY5/fragmento1.bin
-rw-r--r-- 1 root root 167936 Jun 27 04:22 /tmp/tmp.Pq2s2IxJY5/fragmento2.bin
Uniendo los fragmentos en /media/lab/recovered_file.pdf...
Limpiando directorio temporal: /tmp/tmp.Pq2s2IxJY5
Proceso completado. Archivo recuperado en: /media/lab/recovered_file.pdf
pablo@localhost:~/Desktop$ ls /media/lab/
f_01.txt  f_06.txt  f_12.txt  f_17.txt  f_23.txt  f_28.txt  f_33.txt  f_38.txt  f_43.txt  f_48.txt  recovered_file.pdf
f_02.txt  f_07.txt  f_13.txt  f_18.txt  f_24.txt  f_29.txt  f_34.txt  f_39.txt  f_44.txt  f_49.txt
f_03.txt  f_08.txt  f_14.txt  f_19.txt  f_25.txt  f_30.txt  f_35.txt  f_40.txt  f_45.txt  f_50.txt
f_04.txt  f_09.txt  f_15.txt  f_21.txt  f_26.txt  f_31.txt  f_36.txt  f_41.txt  f_46.txt  f_51.txt
f_05.txt  f_11.txt  f_16.txt  f_22.txt  f_27.txt  f_32.txt  f_37.txt  f_42.txt  f_47.txt  f_52.txt
```

Ilustración 35: Ejecución de el script recuperar pdf

Tras recuperar la información se realiza una comparación con el pdf original para ver si se ha recuperado correctamente:

```
pablo@localhost:~/Desktop$ hexdump -C -n 0x75 Plano\ de\ la\ Península\ de\ La\ Magdalena.pdf
00000000  25 50 44 46 2d 31 2e 37  0a 25 e2 e3 cf d3 0a 38  |%PDF-1.7.%.....8|
00000010  20 30 20 6f 62 6a 0a 3c  3c 0a 2f 43 6f 6c 6f 72  | 0 obj.<<./Color|
00000020  53 70 61 63 65 20 2f 44  65 76 69 63 65 47 72 61  |Space /DeviceGra|
00000030  79 0a 2f 53 75 62 74 79  70 65 20 2f 49 6d 61 67  |y./Subtype /Imag|
00000040  65 0a 2f 48 65 69 67 68  74 20 31 32 33 38 0a 2f  |e./Height 1238./|
00000050  54 79 70 65 20 2f 58 4f  62 6a 65 63 74 0a 2f 57  |Type /XObject./W|
00000060  69 64 74 68 20 31 37 32  31 0a 2f 42 69 74 73 50  |idth 1721./BitsP|
00000070  65 72 43 6f 6d                                    |erCom|
00000075
pablo@localhost:~/Desktop$ hexdump -C -n 0x75 recovered_file.pdf
00000000  25 50 44 46 2d 31 2e 37  0a 25 e2 e3 cf d3 0a 38  |%PDF-1.7.%.....8|
00000010  20 30 20 6f 62 6a 0a 3c  3c 0a 2f 43 6f 6c 6f 72  | 0 obj.<<./Color|
00000020  53 70 61 63 65 20 2f 44  65 76 69 63 65 47 72 61  |Space /DeviceGra|
00000030  79 0a 2f 53 75 62 74 79  70 65 20 2f 49 6d 61 67  |y./Subtype /Imag|
00000040  65 0a 2f 48 65 69 67 68  74 20 31 32 33 38 0a 2f  |e./Height 1238./|
00000050  54 79 70 65 20 2f 58 4f  62 6a 65 63 74 0a 2f 57  |Type /XObject./W|
00000060  69 64 74 68 20 31 37 32  31 0a 2f 42 69 74 73 50  |idth 1721./BitsP|
00000070  65 72 43 6f 6d                                    |erCom|
00000075
pablo@localhost:~/Desktop$ ^[[200~diff -s archivo1.txt archivo2.txt
diff: command not found
pablo@localhost:~/Desktop$ diff -s Plano\ de\ la\ Península\ de\ La\ Magdalena.pdf recovered_file.pdf
Files Plano de la Península de La Magdalena.pdf and recovered_file.pdf are identical
pablo@localhost:~/Desktop$
```

Ilustración 36: Comprobación con el pdf original para comprobar si la recuperación se ha hecho correctamente

4 VFAT: nombres largos de fichero

En este capítulo se analizarán las diferencias que se dan entre los archivos dependiendo de el nombre que se les ponga a estos, el como un nombre más largo puede producir

Lo primero que ahy que hacer es crear una partición distinta, en este caso se ha añadido un disco exactamente igual al disco del apartado anterior, pero en esta caso es /dev/sdc y la partición /dev/sdc1

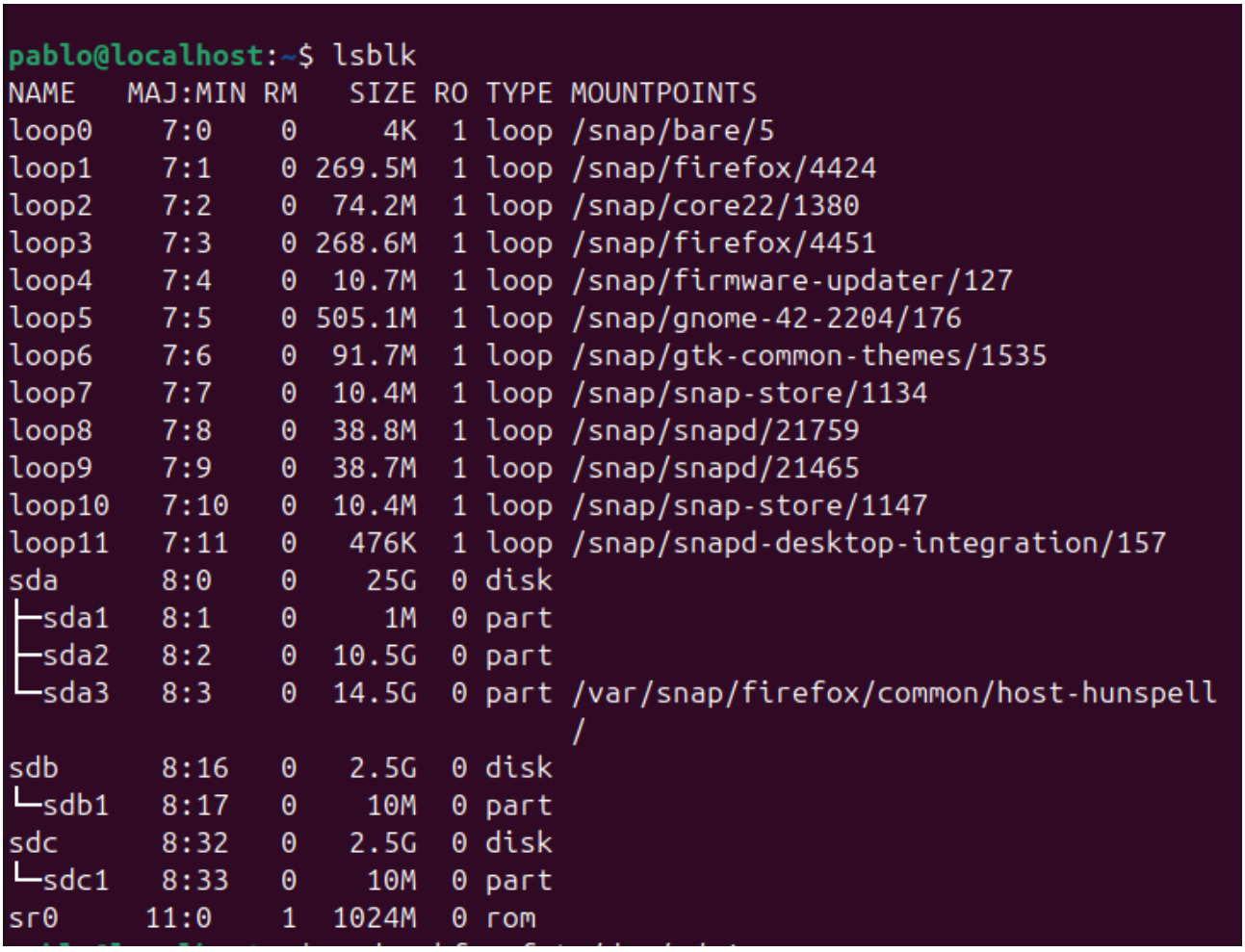


Ilustración 37: lsblk del nuevo disco y partición.

En este caso, s emonta la partición sdc1 en el directorio /media/lab y se crea el directorio /media/lab/NOMBRE. Este será el directorio principal a lo largo de este capítulo.

```
pablo@localhost:/media$ sudo mkfs.vfat /dev/sdc1
mkfs.fat 4.2 (2021-01-31)
pablo@localhost:/media$ sudo mount /dev/sdc1 /media/lab
pablo@localhost:/media$
```

Ilustración 38: lsblk del nuevo disco y partición.

Para poder trabajar y analizar el cómo se organiza la información en este directorio el primer paso es identificar el número de clúster donde se almacena el contenido del directorio NOMBRE, esto se consigue haciendo uso del comando fatcat en la partición.

```
pablo@localhost:/media/lab$ cd NOMBRE/
pablo@localhost:/media/lab/NOMBRE$ ls -l
total 0
pablo@localhost:/media/lab/NOMBRE$ sudo fatcat /dev/sdc1 -l /
[sudo] password for pablo:
Listing path /
Directory cluster: 0
d 27/6/2024 03:29:40  NOMBRE/ (NOMBRE)
            c=3
pablo@localhost:/media/lab/NOMBRE$
```

Ilustración 39: Partición sdc1 siendo FAT y montada.

A continuación se crean 2 archivos de texto cuyo contenido es el mismo, pero con nombre distinto. Para ello se ha programado un script que crea archivos .txt que contienen ”blablabla”pero te pide que le pongas el nombre al archivo a modo de argumento para el programa.

```bash
#!/bin/bash

# Verificar que se haya proporcionado un argumento
if [ $# -ne 1 ]; then
    echo "Uso: $0 <nombre_archivo>"
    exit 1
fi

# Directorio donde se creará el archivo
directorio="/media/lab/NOMBRE"

# Verificar si el directorio existe
if [ ! -d "$directorio" ]; then
    echo "Error: El directorio '$directorio' no existe."
    exit 1
fi

# Obtener el nombre del archivo desde el primer argumento
nombre_archivo="$1"

# Crear el archivo de texto con el contenido "blablabla" en el directorio especificado
echo "blablabla" > "${directorio}/${nombre_archivo}"

# Verificar si la creación del archivo fue exitosa
```

```
if [ $? -eq 0 ]; then
    echo "Archivo '${nombre_archivo}' creado exitosamente en '${directorio}'."
else
    echo "Error al crear el archivo '${nombre_archivo}' en '${directorio}'."
fi
```

Listing 4.1: Script que genera textos iguales de distinto nombre.

Caso de Uso	Descripción
Verificar que se haya proporcionado un argumento	Comprueba que se haya proporcionado exactamente un argumento al ejecutar el script; si no, muestra un mensaje de uso y termina el script.
Definir el directorio de destino	Establece la variable directorio con la ruta donde se creará el archivo (/media/lab/NOMBRE).
Verificar existencia del directorio	Comprueba si el directorio especificado existe; si no, muestra un mensaje de error y termina el script.
Obtener el nombre del archivo desde el argumento	Asigna el primer argumento a la variable nombre_archivo para usarlo como nombre del archivo a crear.
Crear archivo de texto con contenido especificado	Crea un archivo con el nombre proporcionado en el directorio especificado, escribiendo "blablabla" en él.
Verificar la creación del archivo	Comprueba si el archivo se creó exitosamente y muestra un mensaje de confirmación o de error según el resultado.

Ilustración 40: Casos de Uso del Script.

```
pablo@localhost:~/Desktop$ chmod +x crea_texto_nombre.sh
pablo@localhost:~/Desktop$ sudo ./crea_texto_nombre.sh FICHERO.TXT
Archivo 'FICHERO.TXT' creado exitosamente en '/media/lab/NOMBRE'.
pablo@localhost:~/Desktop$ sudo ./crea_texto_nombre.sh archivo.txt
Archivo 'archivo.txt' creado exitosamente en '/media/lab/NOMBRE'.
pablo@localhost:~/Desktop$ ls /media/lab/NOMBRE/
FICHERO.TXT  archivo.txt
pablo@localhost:~/Desktop$
```

Una vez el directorio ya cuenta con contenido que poder analizar, se utiliza el hexdump en el clúster número 3, para ver como se almacenan los archivos que acabamos de crear en NOMBRE.

```
000057f3  00 00 00 00 00 00 00 00  00 00 00 00 00 4e 4f 4d  |.............NOM|
00005803  42 52 45 20 20 20 20 20  10 00 31 b4 1b db 58 db  |BRE     ..1...X.|
00005813  58 00 00 20 23 db 58 03  00 00 00 00 00 00 00 00  |X.. #.X.........|
00005823  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00009ff3  00 00 00 00 00 00 00 00  00 00 00 00 00 2e 20 20  |..............  |
0000a003  20 20 20 20 20 20 20 20  10 00 31 b4 1b db 58 db  |        ..1...X.|
0000a013  58 00 00 b4 1b db 58 03  00 00 00 00 00 2e 2e 20  |X.....X........ |
0000a023  20 20 20 20 20 20 20 20  10 00 31 b4 1b db 58 db  |        ..1...X.|
0000a033  58 00 00 b4 1b db 58 00  00 00 00 00 00 46 49 43  |X.....X......FIC|
0000a043  48 45 52 4f 20 54 58 54  20 00 b5 14 23 db 58 db  |HERO TXT ...#.X.|
0000a053  58 00 00 14 23 db 58 04  00 0a 00 00 00 41 61 00  |X...#.X......Aa.|
0000a063  72 00 63 00 68 00 69 00  0f 00 b3 76 00 6f 00 2e  |r.c.h.i....v.o..|
0000a073  00 74 00 78 00 74 00 00  00 00 00 ff ff 41 52 43  |.t.x.t.......ARC|
0000a083  48 49 56 4f 20 54 58 54  20 00 b0 20 23 db 58 db  |HIVO TXT .. #.X.|
0000a093  58 00 00 20 23 db 58 05  00 0a 00 00 00 00 00 00  |X.. #.X.........|
0000a0a3  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0000a7f3  00 00 00 00 00 00 00 00  00 00 00 00 00 62 6c 61  |.............bla|
0000a803  62 6c 61 62 6c 61 0a 00  00 00 00 00 00 00 00 00  |blabla..........|
0000a813  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0000aff3  00 00 00 00 00 00 00 00  00 00 00 00 00 62 6c 61  |.............bla|
0000b003  62 6c 61 62 6c 61 0a 00  00 00 00 00 00 00 00 00  |blabla..........|
0000b013  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Se puede observar que, en lo que respecta a el área de datos, los dos archivos son idénticos en cuanto a contenido, sin embargo, se ve que en las Tablas FAT, se puede ver como, en el caso del FICHERO.TXT el nombre se guarda tal cual está escrito. Esto no es lo que se ve con el archivo.txt, cuyo nombre ocupa a parte de guardarse como su nombre original en minúscula, y también se guarda convirtiéndolo a Mayúsculas. Almacenamos ahora un fichero con un nombre que sea excesivamente largo:

```
pablo@localhost:~/Desktop$ sudo ./crea_txt_nombre.sh nombre-extremada
mente-largo-que-NO-tiene-sentido-de-lo-LARGO-que-es.TXT
Archivo 'nombre-extremadamente-largo-que-NO-tiene-sentido-de-lo-LARGO
-que-es.TXT' creado exitosamente en '/media/lab/NOMBRE'.
pablo@localhost:~/Desktop$
```

```
0000a0b3  ff ff ff ff ff ff ff 00  00 ff ff ff ff 05 6c 00  |..............l.|
0000a0c3  6f 00 2d 00 4c 00 41 00  0f 00 cb 52 00 47 00 4f  |o.-.L.A....R.G.O|
0000a0d3  00 2d 00 71 00 75 00 00  00 65 00 2d 00 04 65 00  |.-.q.u...e.-..e.|
0000a0e3  2d 00 73 00 65 00 6e 00  0f 00 cb 74 00 69 00 64  |-.s.e.n....t.i.d|
0000a0f3  00 6f 00 2d 00 64 00 00  00 65 00 2d 00 03 6f 00  |.o.-.d...e.-..o.|
0000a103  2d 00 71 00 75 00 65 00  0f 00 cb 2d 00 4e 00 4f  |-.q.u.e....-.N.O|
0000a113  00 2d 00 74 00 69 00 00  00 65 00 6e 00 02 61 00  |.-.t.i...e.n..a.|
0000a123  64 00 61 00 6d 00 65 00  0f 00 cb 6e 00 74 00 65  |d.a.m.e....n.t.e|
0000a133  00 2d 00 6c 00 61 00 00  00 72 00 67 00 01 6e 00  |.-.l.a...r.g..n.|
0000a143  6f 00 6d 00 62 00 72 00  0f 00 cb 65 00 2d 00 65  |o.m.b.r....e.-.e|
0000a153  00 78 00 74 00 72 00 00  00 65 00 6d 00 4e 4f 4d  |.x.t.r...e.m.NOM|
0000a163  42 52 45 7e 31 54 58 54  20 00 5a c3 5d db 58 db  |BRE~1TXT .Z.].X.|
0000a173  58 00 00 c3 5d db 58 06  00 0a 00 00 00 00 00 00  |X...].X.........|
0000a183  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Como se puede observar, en la tabla se alcanza a leer los últimos caracteres del nombre que hemos elegido, pero ¿por qué pasa esto?

En los sistemas de archivos FAT de archivo tradicionales están limitados a un formato conocido como ”8.3”. Esto significa que el nombre del archivo puede tener hasta 8 caracteres, seguido de un punto y una extensión de hasta 3 caracteres. Ejemplos de esto son lo dos primeros archivos de texto, archivo.txt y FICHERO.TXT En las tablas VFAT, que son las que estamos utilizando a lo largo de este trabajo, se aprovechan de este sistema de nombramiento para diseñar dos formas de gestionar estos nombres largos:

Como se explica en el documento de Mechtly et al. [2019]: **Entrada 8.3 estándar:** La primera es resumir el texto para que ocupe un estándar 8.3, un ejemplo sería ThisIsALongFileName.txt se podría resumir a THISIS-1.txt. Para formar nombres en formato 8.3 a partir de nombres largos en sistemas de archivos FAT, se siguen reglas específicas para asegurarse de que el nombre corto sea único y compatible con sistemas más antiguos. Esta estrategia de nombramiento de ficheros se divide en tres pasos:

1. **Truncar el Nombre Base y la Extensión**

   a) **Nombre Base:** Si el nombre del archivo tiene más de 8 caracteres, se trunca a los primeros 6 caracteres y se le añade un sufijo de 1.

   b) Extensión: Si la extensión del archivo tiene más de 3 caracteres, se trunca a los primeros 3 caracteres.

2. **Convertir a Mayúsculas** El nombre base y la extensión se convierten a mayúsculas porque el formato 8.3 no distingue entre mayúsculas y minúsculas.

3. **Manejo de Conflictos** Si el nombre truncado y convertido a 8.3 ya existe en el directorio, se incrementa el sufijo numérico después del tilde ( ) hasta encontrar un nombre único.

Entradas LFN (Long File Name): Para almacenar los nombres de archivo largos, VFAT utiliza múltiples entradas de directorio adicionales, cada una de las cuales almacena una parte del nombre largo del archivo, más concretamente, los últimos caracteres::

Segmentación del nombre largo: Si el nombre del archivo es muy largo, se divide en segmentos de hasta 13 caracteres, que se van numerando secuencialmente y están en orden inverso, desde el final del nombre hasta el principio. **Entradas LFN múltiples:** Para el ejemplo de como funciona la segmentación se hará uso del ejemplo de nombre largo que hemos añadido a /media/lab/NOMBRE. Para esto es necesario conocer la estructura de las entradas LFN, que ocupa 32 Bytes, es esta:

1. Orden (1 byte): Indica la posición de esta entrada en la secuencia. El último fragmento del nombre tiene el bit de alto orden (0x40) establecido.
2. Nombre1 (10 bytes): Primeros 5 caracteres del nombre (cada carácter ocupa 2 bytes).
3. Atributos (1 byte): Siempre 0x0F para LFN.
4. Tipo (1 byte): Siempre 0x00.
5. Checksum (1 byte): Checksum de la entrada 8.3 correspondiente.
6. Nombre2 (12 bytes): Siguientes 6 caracteres del nombre.
7. Primero Cluster (2 bytes): Siempre 0x0000 para LFN. Nombre3 (4 bytes): Últimos 2 caracteres del nombre

Ahora sabiendo la estructura que se debe encontrar se buscan las entradas ayudándonos del hexdump que hemos sacado antes.

```
0000a0c3  6f 00 2d 00 4c 00 41 00  0f 00 cb 52 00 47 00 4f  |o.-.L.A....R.G.O|
0000a0d3  00 2d 00 71 00 75 00 00  00 65 00 2d 00 04 65 00  |.-.q.u...e.-..e.|
```

Ilustración 41: Entrada 1 dice: o-LARGO-que-e

```
0000a0e3  2d 00 73 00 65 00 6e 00  0f 00 cb 74 00 69 00 64  |-.s.e.n....t.i.d|
0000a0f3  00 6f 00 2d 00 64 00 00  00 65 00 2d 00 03 6f 00  |.o.-.d...e.-..o.|
```

Ilustración 42: Entrada 2 dice: -sentido-de

```
0000a103  2d 00 71 00 75 00 65 00  0f 00 cb 2d 00 4e 00 4f  |-.q.u.e....-.N.O|
0000a113  00 2d 00 74 00 69 00 00  00 65 00 6e 00 02 61 00  |.-.t.i...e.n..a.|
```

Ilustración 43: Entrada 3 dice: que-NO-tiene

```
0000a123  64 00 61 00 6d 00 65 00  0f 00 cb 6e 00 74 00 65  |d.a.m.e....n.t.e|
0000a133  00 2d 00 6c 00 61 00 00  00 72 00 67 00 01 6e 00  |.-.l.a...r.g..n.|
```

Ilustración 44: Entrada 4 dice: damente-largo

```
0000a143  6f 00 6d 00 62 00 72 00  0f 00 cb 65 00 2d 00 65  |o.m.b.r....e.-.e|
0000a153  00 78 00 74 00 72 00 00  00 65 00 6d 00 4e 4f 4d  |.x.t.r...e.m.NOM|
```

Ilustración 45: Entrada 5 dice: nombre-extrem

Como se ha explicado antes, las entradas se organizan en orden inverso, el nombre que se le ha dado a el archivo es nombre-extrem-damente-largo-que-NO-tiene-sentido-de-o-largo-que-e

Por lo tanto siguiendo la normativa 8.3 el nombre que recibe truncado seria NOMBRE~1TXT

5 Conclusiones y trabajo futuro

Este trabajo da una visión clara y explicativa de de herramientas y técnicas cruciales en el campo de la ciencia forense de recuperación de datos. Tras habernos basado en gran medida en el libro de Lin y Lagerstrom-Fife [2018].

El principal uso que tiene este trabajo es en el ámbito de la seguridad de la información. En cualquier contexto moderno actual es de suma importancia que la información no pueda ser manipulada, alterada o robada. Conocer cómo se almacenan y ordenan las entradas de directorios y automatizar el cálculo de direcciones de memoria permite verificar la integridad y autenticidad de datos específicos.

Un estudio detallado de cómo se almacenan y gestionan los datos en sistemas de archivos, como lo es este, puede llevar al desarrollo de técnicas más eficientes para la recuperación de datos en un futuro. En situaciones críticas, como la perdida de datos de dispositivos dañados o formateados, las técnicas optimizadas pueden marcar la diferencia entre recuperar información valiosa o perderla.

Así mismo y también relacionado con la protección de datos, el trabajo permite comprobar el funcionamiento de comandos que eliminan información confidencial. Esto abre la puerta a la realización de estudios para mejorar la seguridad de estructuras internas de sistemas de la información.

En otro punto, quizás menos crítico un conocimiento exhaustivo de las estructuras de almacenamiento de la información y de las distintas formas en las que la información se puede almacenar, puede dar lugar al desarrollo de nuevos sistemas de almacenamiento o incluso sistemas operativos nuevos con mayor eficiencia.

El trabajo se ha desarrollado utilizando el sistema operativo Linux por implementar todos los dispositivos como ficheros, que se encuentran en la carpeta '/dev'. No es difícil hacerse con el contenido de la memoria RAM (podemos mirar '/dev/mem') y volcarlo a un fichero binario. En este documento no entramos en el análisis de la memoria ya que implica analizar el contenido.

Algunos sistemas de ficheros (como ext3) apuntan las operaciones que hacen y, en ocasiones, también registran el contenido mismo de los ficheros en un «journal». Incluso se puede configurar este comportamiento al montar el sistema de ficheros. Este journal tampoco se ha revisado a fondo, y se deja como trabajo futuro. Tampoco investigar sistemas de archivos de logs, temporales o la memoria «swap» que cuyo estudio podría dar información sobre aplicaciones utilizadas.

Referencias

W. Chen y C. mei Liu. The analysis and design of linux file system based on computer forensic. In *2010 International Conference On Computer Design and Applications*, volume 2, pages V2–60–V2–64, 2010.
https://ieeexplore.ieee.org/document/5541078

X. Lin y X. Lagerstrom-Fife. *Introductory Computer Forensics*. Springer, Berlin, Germany, 1st edition, 2018.
https://link.springer.com/book/10.1007/978-3-319-92906-2 Available online.

T. Ling. The study of computer forensics on linux. In *2013 International Conference on Computational and Information Sciences*, pages 294–297, 2013.
https://ieeexplore.ieee.org/document/6643000/citations?tabFilter=papers#citations

B. Mechtly, F. Helbert, D. Cox, y Z. Hastings. The visible file system – an application for teaching file system internals. *Washburn University*, pages 24–31, 2019.
https://www.ccsc.org/publications/journals/CPSW2019.pdf#page=24