Lab 5:

ext journals

Forensic Analysis 2023-2024

© Daan Pareit

## Introduction

# Lab concept

This lab is about understanding the metadata structure of ext file systems (block groups, superblock, bitmaps, inode table etc.) and the ext journal. It has a very simple goal: capture the flags (CTF). You are handed over an external disk, which you will examine. Recover the deleted data and find the flags.

# Practicalities

We will use Kali Linux as user environment for analyzing the disk (although any linux distro could do the job). Use the Kali machine from the first lab.

We will work with an additional Virtual Disk (vmdk) which contains (deleted/hidden) data (“the suspect’s disk”). You can find the vmdk at:

* OneDrive: <https://studenthowest-my.sharepoint.com/:f:/g/personal/daan_pareit_howest_be/ErrJ8sYav4tPjpJW8H3kYBoB_b0u1Yqdo96O1Wv8ajehAg>

On kali, we’ll use tools from the Sleuthkit package (TSK). You should already have this installed. If not, you can find the sleuthkit software here: <http://sleuthkit.org/sleuthkit/download.php>

# Learning goals

Knowledge

* ext metadata structure
* superblock, block groups, block bitmap, inode bitmap, inode table
* ext journal structure

Skills

* Perform file recovery on ext3 file systems using the journal
* Find file names of deleted files on ext3/ext4 file systems using inodes
* Use these tools properly: fls, istat, blkcat, file, dumpe2fs, fsstat, xxd, dd, jls, ext3grep, debugfs

## Lab assignments

# Handle with care – Acquisition phase

### The usual good practice

As in previous lab with Windows file systems, you should treat the virtual disk (vmdk) you downloaded as a hard disk from a suspect and should thus handle it forensically correctly. You might want to have a look at *the section “Handle with care – Acquisition phase of external disks” of the first lab*, about how you did this exactly. “Repetition is the mother of learning”, says the proverb.

A reminder of the steps you’ll have to take:

1. Stop/mask the udisks2 service (to prevent automounting)

Open 2 terminals – udevadm monitor + udisksctl monitor

systemctl stop udisks2.service

systemctl mask udisks2.service

1. Physically add the disk (the extra vmdk) to your Kali (adding SCSI disks is hot swappable)
2. Rescan the SCSI bus to discover the new disk

echo "- - -" > /sys/class/scsi\_host/host2/scan (as root)

1. (optional: read disk metadata with udevadm/hdparm)
2. Create an MD5 hash of the suspect’s disk with md5sum.

sudo md5sum /dev/sdb1

74874b8fac2610520f9e7417b4162cff /dev/sdb1

sudo md5sum /dev/sdb

e35d1c2037e62351f6d15a3f55b575d0 /dev/sdb

1. Make a backup image of *the full disk* using dd (e.g. sdb-backup.img).

sudo dd if=/dev/sdb of=sdb-backup.img status=progress

1. Create an MD5 hash for your backup. Verify that suspect’s disk and the backup have the same.

sudo md5sum /dev/sdb sdb-backup.img

e35d1c2037e62351f6d15a3f55b575d0 /dev/sdb

e35d1c2037e62351f6d15a3f55b575d0 sdb-backup.img

1. Disconnect the disk in linux from yourls device list

echo 1 > /sys/block/sdb/device/delete (as root)

1. Physically detach the disk (the extra vmdk) from your Kali
2. (optional: restart udisks2 service)

sudo systemctl unmask udisks2.service

sudo systemctl restart udisks2.service

You should not mount the disk itself, because you could alter its data. For the next sections you will always need to perform your investigation on your forensic backup. ***You thus shouldn’t use /dev/sdb in any of the following steps (as it should be disconnected with the steps above) but use your image file instead!***

And again, you can work on the backup image directly (with offsets to access the partitions therein) or create a loopback device to work with (losetup -fPr *imagefile* --show).   
Note: with losetup -l, you can then see which files are made available as loopback block devices.

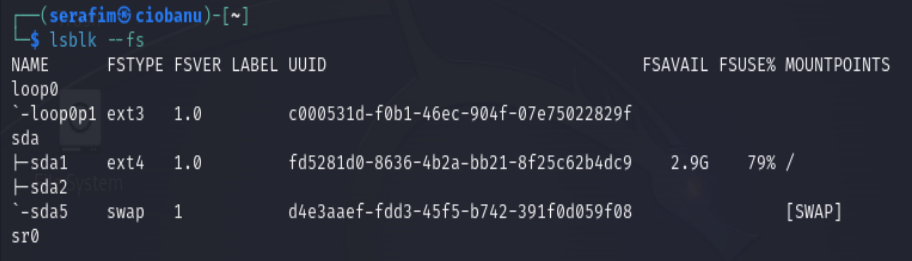
# Analysis of the suspect’s disk

### Disk layout analysis

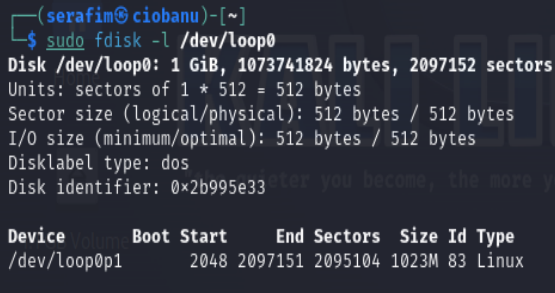
First, have a look at the disk layout to identify the partitions (cfr previous labs) and the file systems they are formatted with:

* Is there any substantial amount of free space (neglect the first free sectors)? How many partitions are there? What file system are they formatted with?  
  (tools: fdisk, parted, lsblk)

lsblk —fs

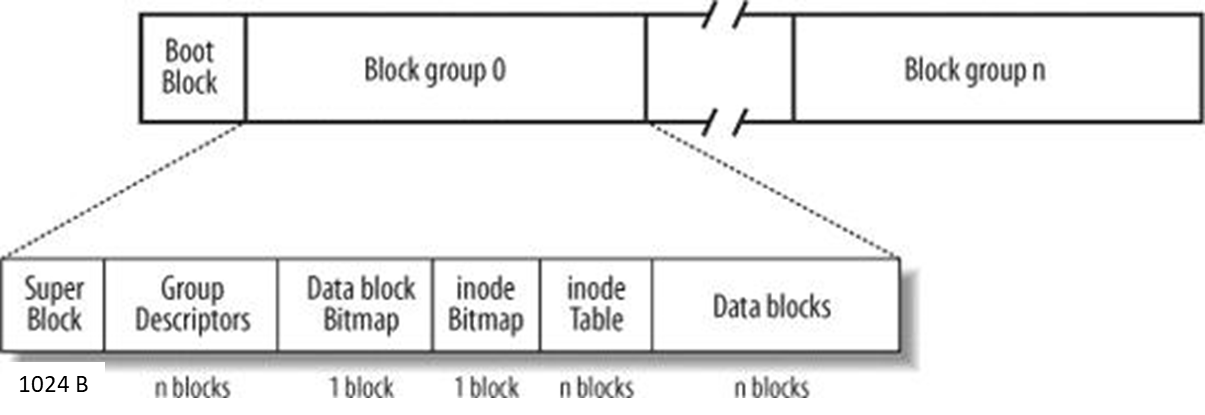


sudo fdisk –l /dev/loop0



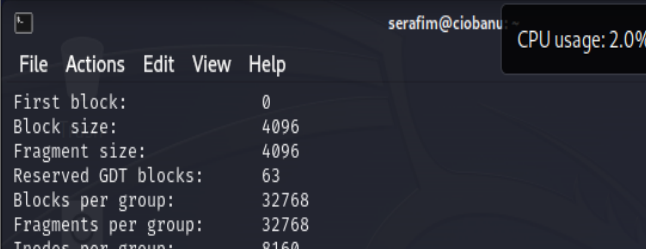
### ext metadata

Before starting to search for deleted data, let’s first get familiar with the ext metadata structures: where are inodes saved on disk, what block groups are present, etc. Keep the figure below in mind, when answering the questions below.



* Inspect with dumpe2fs the block size that is used by the file system on the suspect’s disk.

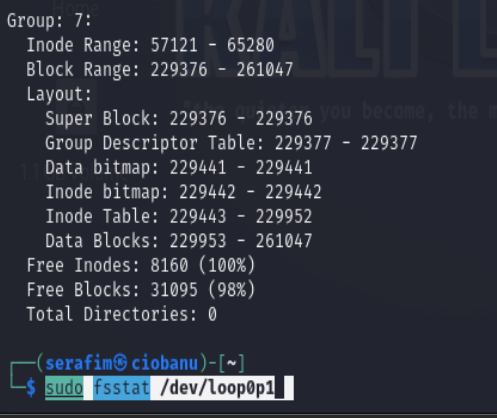
sudo dumpe2fs /dev/loop0p1 | less



4096

* Inspect the file system structure using dumpe2fs or fsstat. The last part of the output shows information about each block group. How many block groups are there in this file system?

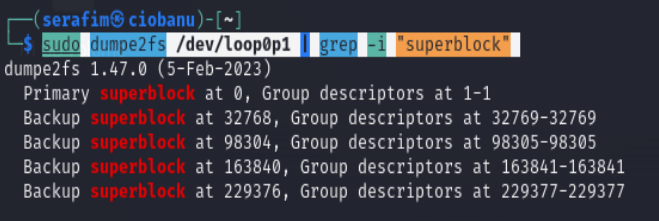
sudo fsstat /dev/loop0p1



The last one is number 7, but we also start from 0, hence I guess the total is 8.

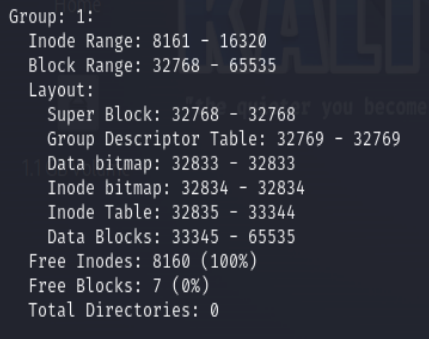
* Which block groups contain a (copy of the) superblock?

sudo dumpe2fs /dev/loop0p1 | grep -i "superblock"



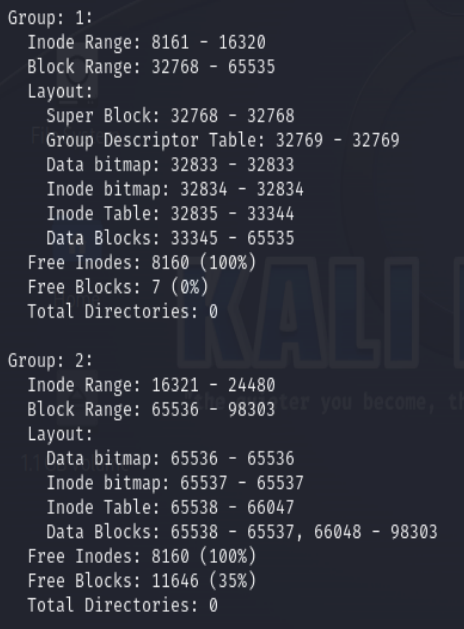
Group 0, 1, 3, 5, 7 ( funny, each 2nd )

* Each block group has a ‘data/block bitmap’ and an ‘inode’ bitmap. How many blocks are each of these in size?



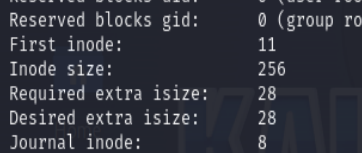
According to the information from the **fsstat**, we can see that the data and indoe bitmap take 1 block each

* How many blocks are there in each block group (except for the last one)?   
  Note: this matches indeed the calculation in the lecture slides, based on the block size and the size of the block bitmap.



To calculate it for example for the Group 1, we can check the block range (32768 - 65535), and then check the next block range of Group 2. We can do 65536 - 32768 = 32768.

* Inspect with dumpe2fs the size of one inode. How many inodes thus fit within one block of this file system?

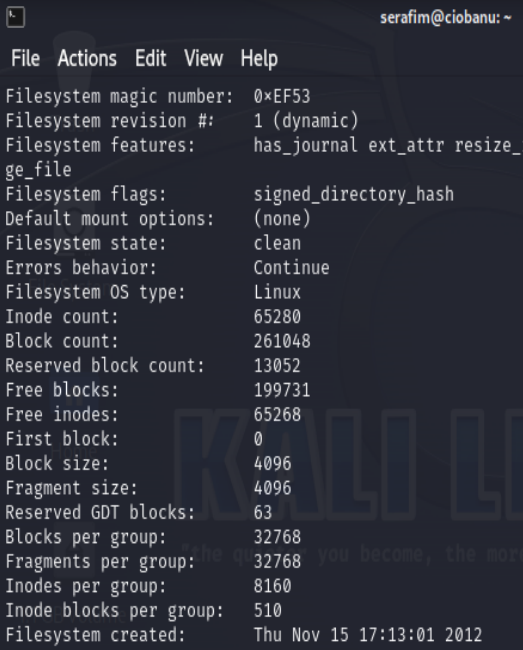


I can see that the tool says 256 inode size

I googled, and it says that I need to divide the block size by the inode size, and I will get the number of inodes per block.

Hence: 4096 / 256 = 16 inodes per block

* Inspect with dumpe2fs or fsstat the number of inodes per block group and the number of blocks that are used to save these inodes per block group.   
  Note that their ratio will match your calculation in the previous question.

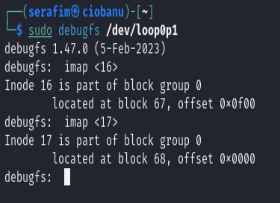


number of inodes per block group = 8160

number of blocks that are used to save = 510 (8160 / 16)

I am not really sure which is which from what I am asked

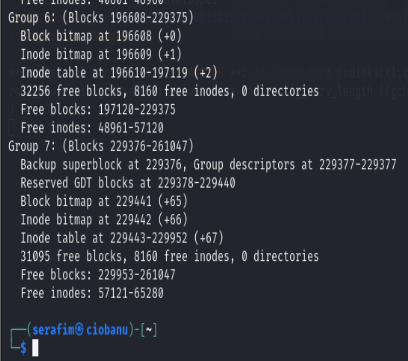
* You can verify with the ‘imap <inode>’ command within debugfs /dev/loop0p1 indeed which inodes are located on the same block.



I really do not understand what am I supposed to look for, because it makes no sense to me.

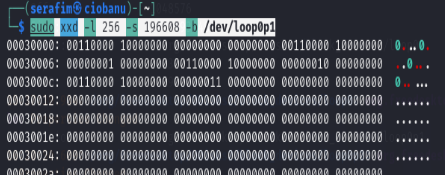
Am I supposed to look for inodes located on one block group, or what?

* Again, use dumpe2fs or fsstat and have a look at block group 6. What is the block number of the block bitmap in this group? Use blkcat and xxd to inspect the content of this block. How many blocks are indicated to be ‘allocated’, or simply put: ‘in use’ (i.e. how many bits are set to 1)?   
  Note that there’s no data saved within this block group yet. The number of allocated blocks will thus be the sum of only the blocks used by the inode bitmap, the block bitmap and the inode table.



The block number of the block bitmap is 196608 as far as I can see

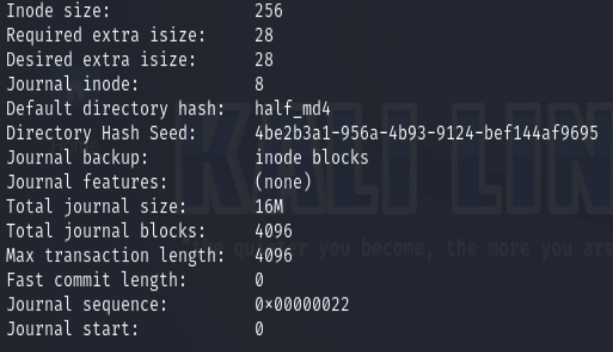
sudo xxd -l 256 -s 196608 -b /dev/loop0p1



As I can understand, there are 16 inodes being used, because they are set to 1

### About the journal itself

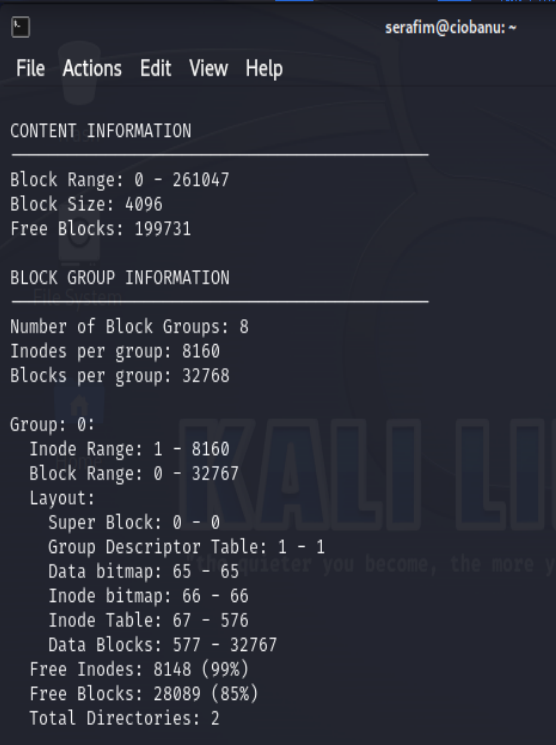
* Again, use dumpe2fs and have a look at the information about the journal. What is saved in the journal? Metadata and/or actual data content?



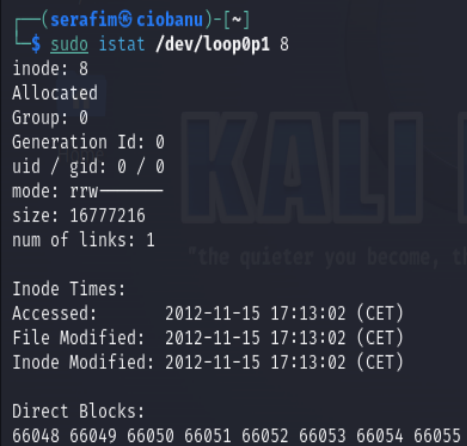
From what I see, I guess it stores only metadata

* What inode number is used for the journal? In what block group is it thus located?

It has inode number 8, and thus it is stored in the block group 0, because it has a inode range from 1-8160



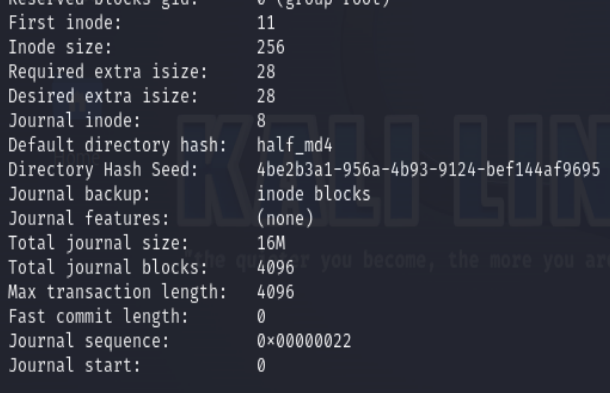
* What is the first data block which is used by the journal? (istat)



It is 66048? Because it is shown as the first block in the list

* How many blocks are being used for the journal? What is its size?

4096



### Recovery of deleted files on ext3 using journal information

The partition has three hidden flags. But as it is ext3… deletion is destructive for the inodes.   
On the bright side: if you find the journal, maybe you have a trace to the *logs* of the delete operation.

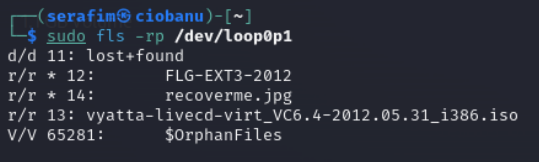
* Find the flags in the ext3 partition
* Be able to say in which inode and in which data blocks the flags are found
* Be able to say in which journal transactions and journal blocks you found this information

Hints

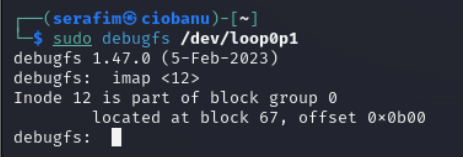
* Finding block of an inode: **debugfs**
* Finding and analysing the journal: **jls, ext3grep**
* Retrieving data: copy blocks using **blkcat** or **dd**.

**First one**

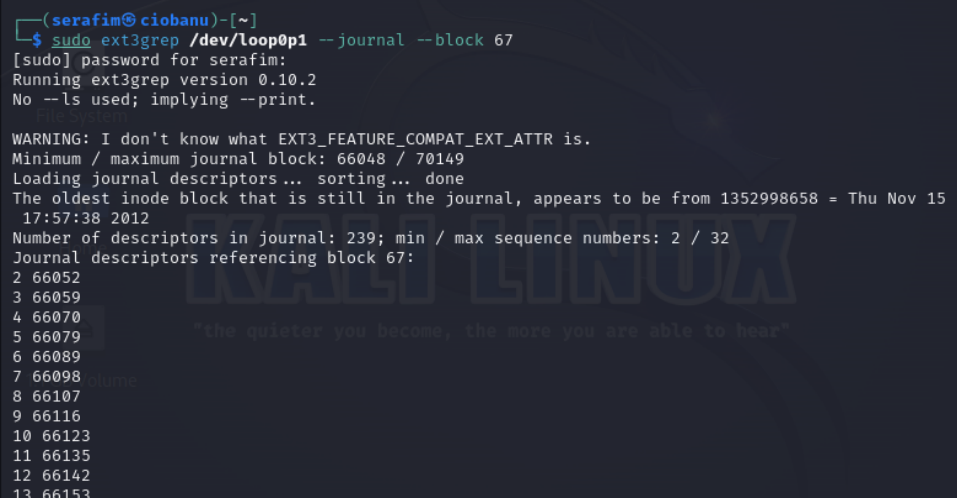
First what I want, is to find the inode related to a deleted file



Then I need the blocks related to it?



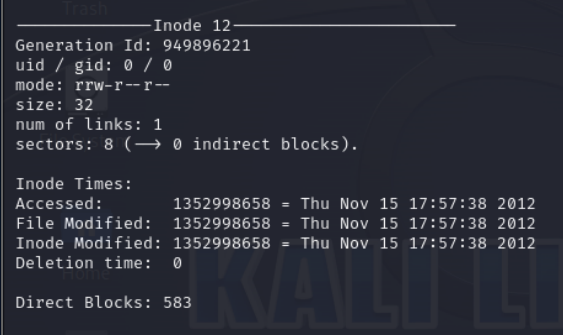
We get a bunch of data blocks where it could be?



And then we want to read the values of a block itself, which was related from the previous command? I just decided to try the first one of them

sudo ext3grep /dev/loop0p1 --block 66052

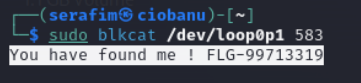
Then we have a bunch of entries, which point to different blocks themselves (or not really, because in the screenshot we can see inode 12 and where it points to)



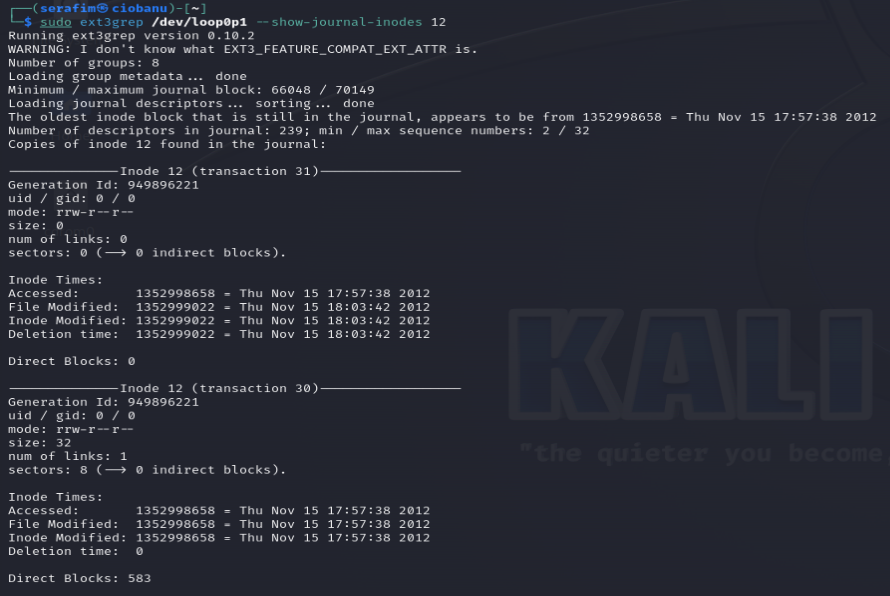
So then, if we try and find the actual content of the file, we will do:

sudo blkcat /dev/loop0p1 583

And in this case – we get the flag (You have found me ! FLG-99713319)

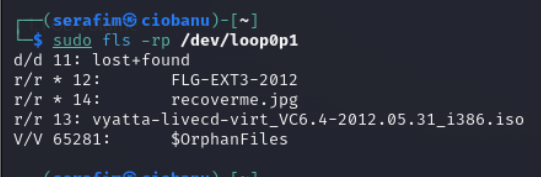


We can also make sure to see the transaction thtat contains information regarding that inode

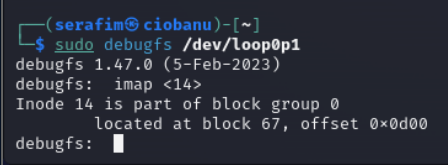


**Second one:**

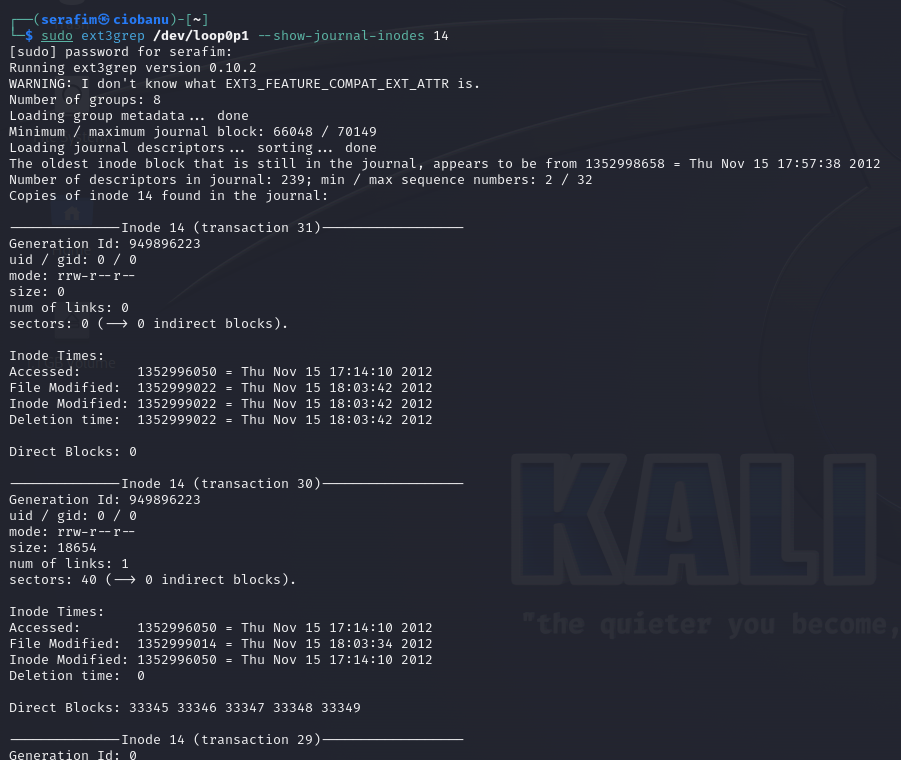
Same, lets look for the next file



Then look where it points to



I found out along with other students, that there is also a way to view information regarding



Here we actually get the information regarding where the blocks of the image is

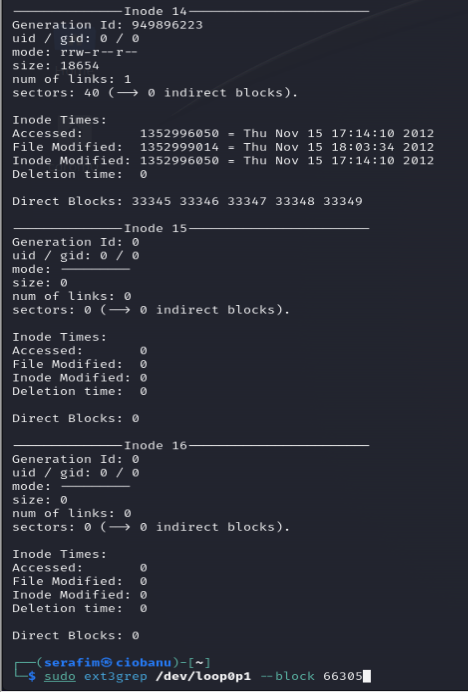
Also, to check for the blocks, after we do the command:

sudo ext3grep /dev/loop0p1 —block 67 —journal

We get the bunch of blocks that are related to it, right?

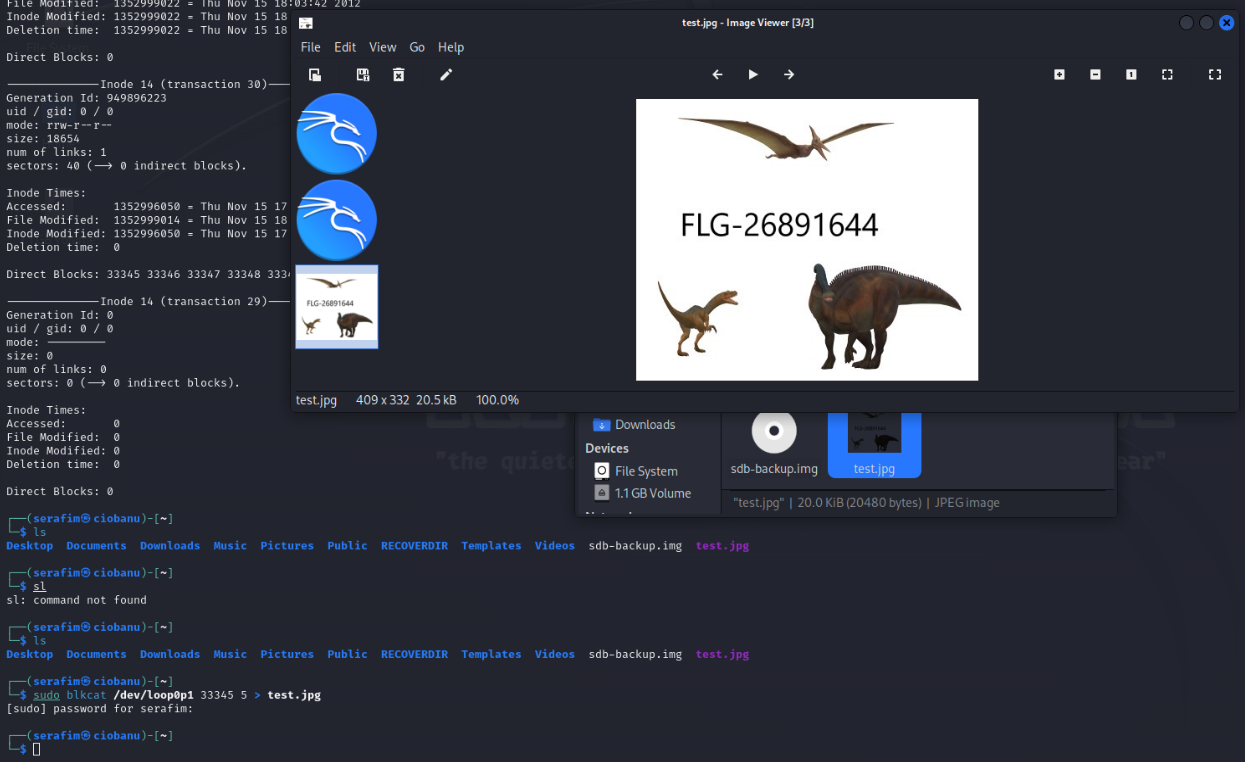
Then what we can do, is check each one of those blocks, and see which one contains information (direct blocks) for the inode 14 itself.

We can do it manually, or by writing some script



Then, to recover it, what we can do, is try and capture the information from the blocks using the blkcat

sudo blkcat /dev/loop0p1 33345 5 > test.jpg

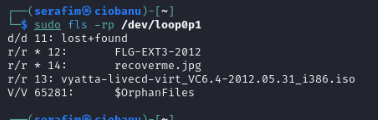


So you captivate information with the blocks into a single block of image (because the inode said that it is an image), and then we get the flag (FLG-26891644)

So the inode is probably 14 in this case, and the blocks are 33345 + 5 (last is 33349)

**Third one:**

The third one appears to be the name of a file itself (FLG-EXT3-2012)



One of the flags starts with “FLG-2” . On Leho, upload a screenshot of the tool which reveals what block(s) contain the actual flag. (Not a screenshot of the flag itself but about where to find it.)

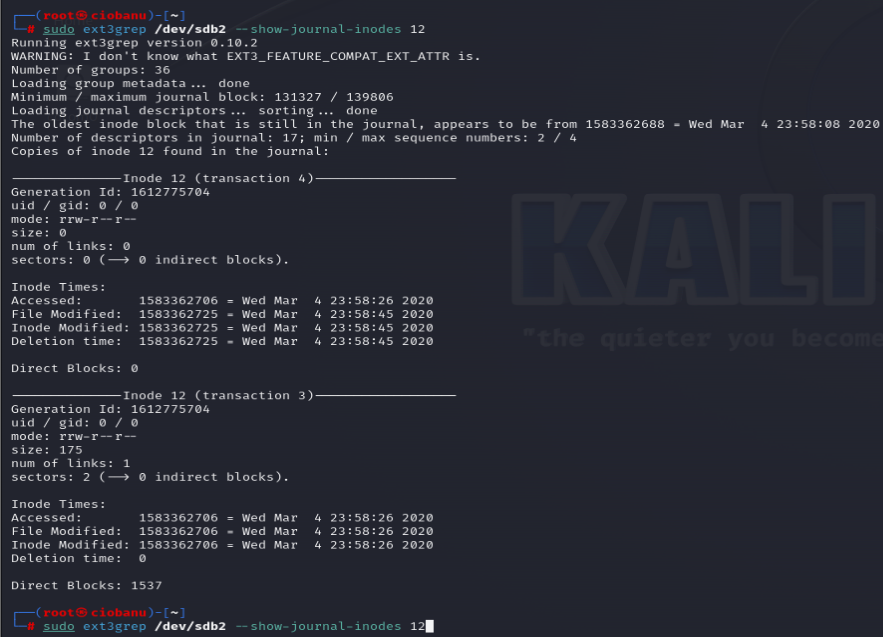
# Re-inspect the disk of previous lab

Now that you’re familiar with the journal, let’s reflect on the previous lab. There, we also had an ext3 and an ext4 partition where we couldn’t identify the data associated with the deleted files.

sudo losetup -fPr sdb-backup-lab4.img --show

* Now try using the journal on the ext3 partition to find the data associated with the deleted file. What block number is the data located in?

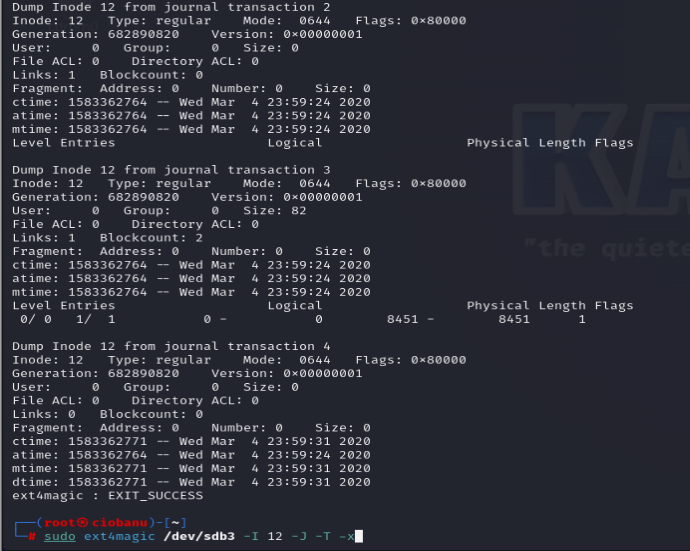
sudo ext3grep /dev/sdb2 --show-journal-inodes 12 (or loop0p2)



Direct Blocks: 1537

* For the ext4 partition as well, find the data associated with the deleted file. Here, ext3grep might fail because it’s not aware of all ext4 features. Use ext4magic instead with similar functionality for ext4. You’ll need I/J/T/x options. What block number is the data located in?

sudo ext4magic /dev/sdb3 -I 12 -J -T -x



Block number: 8451