Recupero

By Group 1

Table of Contents

Abstract....3
Introduction....3
    Use Case....3
Similar Tools....3
EXT2....3
    File Deletion....4
Implementation....5
Challenges....6
    Deletion Problems....6
    Too Much Customization....7
Conclusion....7

Illustration Index

Illustration 1: Ext2 Layout....4
Illustration 2: Inode before delete....4
Illustration 3: Inode after Delete....5

Abstract

Recupero's main purpose is to recover deleted files on the ext2 file system, however Recupero also allows the user to do many things on the ext2 file system. Something unique about Recupero, is that is lets the user view the inner working of the file system. For example you can view the data and inode bitmaps. This feature is unique and we couldn't find this on any other tools.

Introduction

Our area of focus was disk forensics. In computer forensics, before investigators can start analyzing the disk they must first a create an image of the disks in the system. These files are representative of the entire disk. The investigators then use tools to analyze the image. There are various levels of these disks, high and low. Most tools that we know about work at the high level, meaning they try to work with the existing file system on the image.

There are many cases were there is a person being investigated and they have been doing something that they don't want other people knowing about. They tend to delete files that would incriminate them of some wrong doing. Little do they know a lot of the time those "deleted" files aren't completely gone. That is where Recupero comes in.

Use Case

The use case for Recupero is when you have an image of a disk and want to see if there are any deleted files on it. Of course Recupero is limited to the ext2 file system, which limits its usefulness, but since it's technically supported by distributions there is a chance of running into the ext2 file system in the real world.

Similar Tools

There are tools already developed that recovers files on the ext2 file system. The main tool that we came across was called e2undel. This program has one purpose, and that is to recover deleted files in the ext2 file system. Another program with similar functionality is Sleuthkit. Sleuthkit has the advantage over Recupero though, Sleuthkit has support for multiple file system types and more convenient tools. For example Sleuthkit allows the user to traverse directories of an image.

EXT2

Ext2 or the extended file system 2 was made in 1993 to replace the extended file system and in 2001 was "replaced" by ext3. All of the ext file systems are similar in how they work and

basically as time went on features were added to give the file system more capabilities. Even today you can take an ext4 file system and easily convert it to ext2 or ext3 file system.

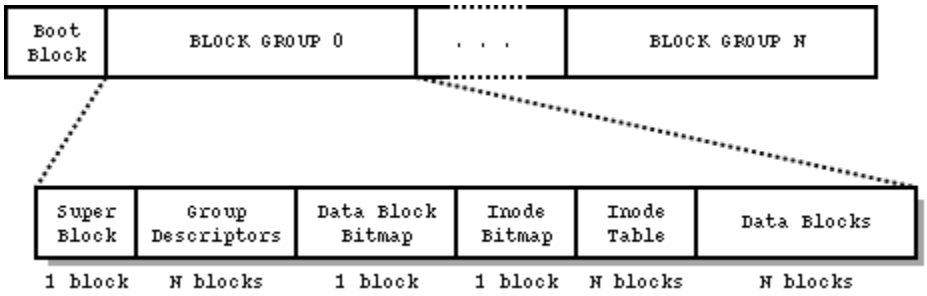

Illustration 1: Ext2 Layout

The layout of the ext2 file system is shown above. The file system is divided into block groups where each block group has a super block, group descriptors, data bitmap, inode bitmap, inode table, and data blocks. A super block stores all relevant information for reading and writing to the rest of the file system. The primary super block is stored after the boot block (which isn't used in ext2) which is 1024 bytes into the file system. Next are the group descriptors, each group descriptor tells you were the bitmaps and inode tables are within each block group. These values are given in blocks that can be easily indexed to when reading the file system. In a later revision of the ext2 file system (revision 1) the super block and group descriptors are only in blocks 0, 1, 3, 5 then powers of 3 and 5. This was to save space as the file systems got larger. The data block bitmap and the inode bitmap tell the file system if a specific inode/data block is free or not. Then lastly the data blocks, this is where the contents of files are stored.

File Deletion

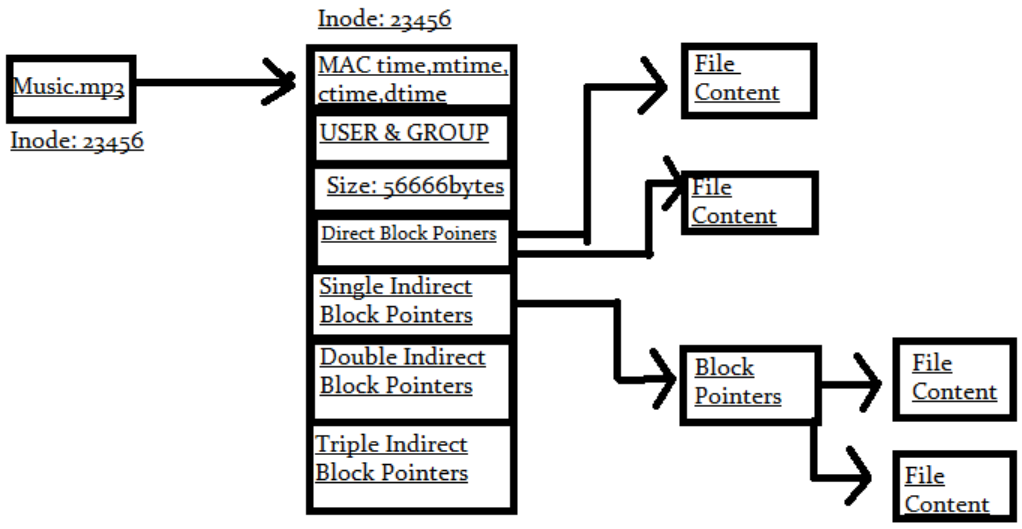

Illustration 2: Inode before delete

When a file is deleted a few things happen, first is that the directory's entry that has the file name and inode of the file becomes invalidated. This means that the original file name is not recoverable. Then the inode is marked as free along with all the data blocks that the inode points to.

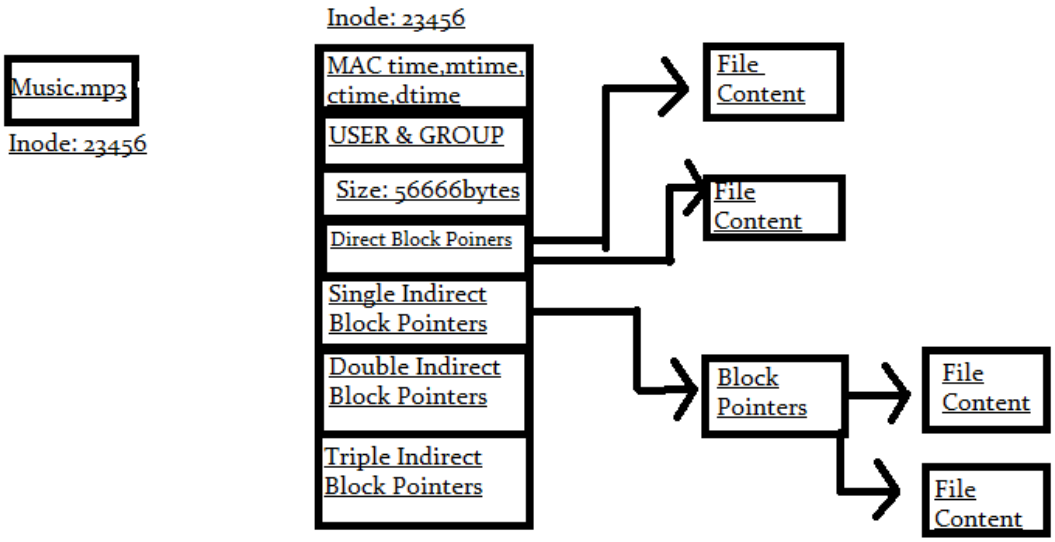

Illustration 3: Inode after Delete

But the most important part is that the inode itself still points to all of the data that it did before. Meaning if we can find this inode we can recover the file. There are couple potential problems though, the inode could get overwritten before the file system get unmounted and imaged, or any of the data blocks that the inode points to could get overwritten.

Implementation

We wanted Recupero to work in two modes, one with an interactive prompt where the user could interact with the program more than normal, and a single command mode. The single command mode would allow Recupero to be scriptable which is the main advantage of the command line over GUIs so we didn't want to miss out on this feature. To accomplish this, we used the number of parameters passed to the program at start to determine the mode, and based on the mode we sent the parameters to either the interactive prompt or try to parse the input right then.

In the actual implementation of Recupero we used the ext2_fs.h which defines all the structs used in the ext2 file system. We got this header file from the Sleuthkit git repo, we used this one instead of the original in the Linux kernel repo because the original header file had multiple dependencies that the Sleuthkit header file already resolved for us.

The program flow is relativity straight forward, for any command (except recover super block) we first read the primary super block. For this we use the ext2_super_block struct and get the most necessary information for the rest of the processing. This includes the block size, block group count, and number of group descriptors. We also check the magic number in the super block to ensure that the file system is actually ext2. Note: we do end up storing storing the entire super block because it has everything needed for processing. This is great because we ran into a couple of instances where we needed a variable we didn't think about and it was already stored in the super block. We then read the group descriptors, using the struct ext2_group_desc we can easily extract all the information we need. This information includes where bitmaps are, and the locations of the inode tables. From here we then execute the specific function that the user wants.

Since recovering deleted files is our program core functionality we have to find deleted files, but how? We came up with two ways of finding deleted files, the first was using the bitmap to check every inode that was free. If there was “good” information in the inode it’s probably recoverable. The problem with is method was how do we determine a “good” inode? The second method we came up with was going through all the inodes and looking for inodes that have a deleted time in them. Normal inodes have this time set to zero, but in deleted inodes this value is non-zero!

The coolest feature that we implemented is when the user views what files are recoverable, it shows a warning if we can determine that the file is corrupt. We do this by checking to see if the data blocks that an inode points to are still free. Be warned, just because this warning isn’t shown doesn’t mean that the recovered file isn’t going to be corrupt. This is because another file could have overwritten that block, but then later got deleted and freed that block.

Challenges

We encountered a few challenges during this project. There were some of the normal problems, communication, planning, and coordination. To solve the problem of communication, our group primarily communicated through Slack. For planning and coordination we had a design document that we were loosely following.

Deletion Problems

By far the biggest problem we encountered was by the time we could read the super block, group descriptors and find deleted inodes we noticed that unlike the documentation for ext2 the inode information was getting zeroed out when it got deleted! This broke our entire project. After investigation we theorize that ext2 is still “supported”, but what really happens is that its an ext4 file system with various flags switch so it’s suppose to act like an ext2 file system. There are clearly inconsistencies with the real ext2 and what we were seeing.

Knowing that at one point programs like e2undel worked, we figured that real ext2 file systems were used in older operating systems. So using the power of the internet we installed one of the oldest versions of Ubuntu we could find, Ubuntu 6.06.2. Using Sleuthkit for verification, we found that this version of Ubuntu treated ext2 correctly, but developing on this platform would be difficult because the repos no longer worked. Then we tried the oldest version that still had repos in place, Ubuntu 12.04. Thankfully this version of Ubuntu also treated ext2 correctly. Knowing this, Ubuntu 12.04 then became our new development target.

Too Much Customization

When making the ext2 file system using mkfs.ex2 there are a lot of options to use. Most of them are the defaults that generally aren't displayed well in the mke2fs configuration file. Some examples of how this was a problem are block size. When indexing, we assumed that certain properties were certain sizes based on the block size. We then hard coded values in but then noticed inconsistencies not working on different block sizes. Another example, inode size. When trying to locate inodes we needed to index in to read them. But this index changes based on the inode size, which the struct of which is 128 bytes, but the default size is 256 to accommodate the ext4 inode. The solution to this problem was to use values stored in the super block instead of hard coding values and being aware of the defaults.

Conclusion

The main goal of this project for us was to learn about files systems in general because ideas from one file system can apply to other file systems. I would say that we have accomplished this goal. Besides learning about files systems, none of the group members knew a lot of c++ which the project was written in. So we were learning c++ as we created the project, that is why there are a lot of inconsistent styles throughout the program, because we were learning as we went and didn't really know the best practices. Other technologies that we used were git (which we mostly knew the basics of), and make. We only talked about makefiles very briefly in other classes, and makefiles are generally the best way to "make" programs. So we had to learn that as well. In the end we had a lot of fun on this project and learned a lot.