João Miguel Nunes Veloso

Degree in Computer Science and Engineering

Automated support tool for forensics investigation on hard disk images

Dissertation plan submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: Francisco Loureiro, Chief Operations Officer, PDM

Co-adviser: João Leitão, Assistant Professor, NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Febuary, 2020

Abstract

The advent of computers and the massification of mass storage has led to the generation of large volumes of data by individuals. This, in turn, created a challenge for the world's law enforcement agencies, who are many times faced with the need to look, in a timely manner, at someone's hard drive to find offensive multimedia contents, such as child abuse digital proof. However, the large amount of data found in captured laptops of suspects turns this task unfeasible, unpractical, and very time-consuming since the investigator has limited time and resources.

When we take into consideration that a suspect might be from a foreign country, the time window available for a criminal investigation is even shorter. Furthermore, some criminals might have some technical knowledge and might try to hide evidence, making the task of the investigator even more daunting.

In this thesis we aim at developing a new system that will assist criminal investigators in looking for evidence on images of hard drives extracted from the devices of suspects. In this work we assume that the content on the hard disk images is not encrypted and we will focus on child abuse multimedia content, although the platform will be designed to be extensible to handle other types of content in the future. The tool will automatically index the contents of the hard drive, and look for potential evidence that is important to the criminal investigators. This tool will be able to help not only police departments but several other organizations such as the INHOPE (Internet Hotline Providers in Europe) and NGOs fighting for children.

Keywords: Digital Forensic, Forensic Science, Picture Triage, Machine learning, Searching

Resumo

A revolução tecnológica das últimas décadas trouxe com ela a massificação do armazenamento em massa, isto levou a uma enorme criação de dados por parte dos indivíduos. Consequentemente, criou um desafio para os investigadores, que muitas vezes se deparam com a necessidade de analisar, em tempo útil o disco rígido de um suspeito para encontrar conteúdos multimédia ofensivos, como por exemplo conteúdos digitais de abuso infantil. No entanto, a grande quantidade dos dados encontrados em computadores capturados de suspeitos torna esta tarefa inviável, impraticável e muito demorada, uma vez que o investigador tem tempo e recursos limitados.

Quando temos em consideração que um suspeito pode ser de um país estrangeiro, a janela de tempo disponível para uma investigação criminal é ainda mais curta, em Portugal possuem apenas 48 horas para apresentar uma acusação. Infelizmente, este não é o único problema que os investigadores enfrentam, alguns criminosos possuem conhecimento técnico e podem tentar esconder provas, tornando a tarefa do investigador ainda mais desencorajadora.

Nesta tese, pretendemos desenvolver um novo sistema que ajudará os investigadores forenses na procura de provas em imagens de discos rígidos extraídas dos dispositivos dos suspeitos. Neste trabalho assumimos que o conteúdo das imagens dos discos rígidos não se encontra encriptada, o nosso foco será em conteúdo multimédia relacionado com abuso infantil, embora a plataforma possa ser estendida para lidar com outros tipos de conteúdo no futuro. A ferramenta indexará automaticamente o conteúdo do disco rígido e procurará potenciais evidências que sejam importantes para os investigadores criminais. Esta ferramenta será capaz de ajudar não apenas os departamentos policiais, mas várias outras organizações como a INHOPE (Internet Hotline Providers in Europe) e ONGs que lutam contra este tipo de material.

Palavras-chave: Forense Digital, Ciência Forense, Triagem de Imagens, Machine learning, Pesquisa

Contents

List of Figures **ix**

List of Tables **xi**

Acronyms **xiii**

1 Introduction **1**

1.1 Context . . . 1

1.2 Motivation . . . 1

1.3 Expected Outcome . . . 2

1.4 Document Structure . . . 2

2 Related Work **3**

2.1 Digital forensics . . . 4

2.2 Architecture . . . 5

2.2.1 Memory . . . 8

2.2.2 Disk . . . 9

2.2.3 Discussion . . . 9

2.3 File system . . . 9

2.3.1 New Technologies File System . . . 10

2.3.2 Hierarchical File System Plus . . . 12

2.3.3 Ext3 Concepts and Analysis . . . 15

2.3.4 Ext4 . . . 18

2.3.5 FAT Concepts and Analysis . . . 18

2.3.6 File system in Userspace . . . 21

2.3.7 Discussion . . . 22

2.4 Machine Learning Techniques . . . 22

2.4.1 Not Safe For Work . . . 23

2.4.2 Age Group Recognition . . . 24

2.4.3 Discussion . . . 24

2.5 Search engine indexing . . . 27

2.5.1 Suffix tree . . . 27

2.5.2 Inverted index . . . 28

2.5.3 N-gram index 28
2.5.4 Image indexing 28
2.5.5 Discussion 28

3 Planning **31**
3.1 Overview 31
3.2 Main task 32
3.3 Design principles 32
3.4 Calendarization 33
3.5 Tools 34
3.6 Evaluation method 35

Bibliography **37**

List of Figures

2.1 Traditional Digital Forensics process 5

2.2 XML-based indexing and querying for digital forensics (XIRAF) framework architecture [2] 6

2.3 HANSKEN Modular view [5] 6

2.4 Simplified initial Architecture 8

2.5 MFT entry [23] 11

2.6 MFT Boot Sector [23] 11

2.7 Example of the MFT entry [23] 12

2.8 Layout of Ext [23] 16

2.9 Layout of a sample block group [23] 17

2.10 Relationship of FAT structure [23] 19

2.11 Physical layout of a FAT file system [23] 19

2.12 FAT file system layout [23] 20

2.13 Filesystem in Userspace (FUSE) high-level architecture [36] 22

3.1 Detailed Architecture 32

3.2 timeline of the upcoming months 34

List of Tables

2.1 Hierarchical File System Plus (HFS Plus) new features [26] 12
2.2 HFS Plus Volume Header field and there meaning [26] 14
2.3 Results of Not safe for work (NSFW) proposed classifiers[44] 25
2.4 Average performance over NSFW dataset 26
2.5 Implementation of a Suffix tree 27

3.1 Chronograph 33

Acronyms

BLOB Binary large object.

CNN Convolutional neural network.

DEX Deep EXpectation.

FUSE Filesystem in Userspace.

GUI Graphical user interface.

HDD Hard Disk Drive.

HFS Hierarchical File System.

HFS Plus Hierarchical File System Plus.

HTTP Hypertext Transfer Protocol.

MBR Master Boot Record.

MFT Master File Table.

NSFW Not safe for work.

NTFS New Technologies File System.

OS Operating system.

RAM Random Access Memory.

ROM Read-only memory.

SSD Solid State Drive.

XIRAF XML-based indexing and querying for digital forensics.

XML Extensible Markup Language.

Chapter 1
Introduction

The only impossible journey is
the one you never begin.

Anthony Robbins

1.1 Context

The current technological revolution in our time is changing our lives, making them more comfortable, more connected, and digital. However, criminals have also started to use more of these technologies to their own activities. Domo, a business cloud service analyst, estimates that more than 2.5 quintillion bytes of data are created every single day, and forecast that this estimate will grow even more, they also predict that by 2020 1.7MB of data will be created every second for every person on earth[1].

This has made the work of law enforcement agents a constant battle to search and filter increasing amounts of data particularly when looking for evidences on drives that were apprehended from suspects. This is, in part due to the decrease of storage cost and the increasing of crimes committed using digital devices.

In this work, we describe a novel approach towards processing, managing, filtering, and querying from disk images obtained from potential suspects and presenting them in a way that is simple for law enforcers to use and still locate evidence in a timely manner.

1.2 Motivation

Every day more and more crimes are committed using computers, they are also getting increasingly sophisticated. This poses a real challenge for law enforcement professionals to keep up.

Upon seizing a suspect hard drive detectives and analysts must guarantee forensic integrity, so they create forensic copies of the digital devices. These images are byte by byte copies of the original.

In the next phase, called the extraction process, they inevitably fall short, because there are not many adequate software solutions to assist in this process, there are many software to choose from, the existing ones require significant technical knowledge, and usually are quite expensive.

The ideal solution would be simple to use and free, here is where our work intents to fill in the gap. Our main goal is to provide free software that can look through disk images and process them in a scalable way, despite the high volumes of digital material. This process must happen in useful time, since suspects cannot be held for an indefinite amount of time. Additionally, it should maximize the trace coverage, meaning the investigator can trace back the evidence and prove it was not planted. Finally, our solution should minimize the necessity of specialized technicians, being this restriction motivated by limited budget of law enforcement agencies.

1.3 Expected Outcome

This work aims to develop and make available a new tool for the forensics community to automate the analysis and data extraction from disk images.

We expect to create and deliver a novel solution that can extract files from the pictures, both present and deleted. In particular, in this work we will ensure that, when an image analysed, it should determine, with a confidence interval if it contains child pornographic.

To give back to the community, help law enforcement agency's and since this work builds on the work of others, the source code with be made available through GitHub.

The developed solution will be tested in a real-world scenario in conjunction with a local law enforcement agency. This will make the solution more stable, reliable, and allow to determine its accuracy.

1.4 Document Structure

The remainder of the document is structured as follows:

- **Chapter 2** describes the Related Work that was studied to elaborate this work. Focusing on underlying systems that manage relevant data and how they operate.
- **Chapter 3** provides a detailed planning structure for the upcoming months, time distribution through the various phases and how the final soluction will be evaluated.

Chapter 2

Related Work

Great things in business are never done by one person. They're done by a team of people..

Steve Jobs

This chapter presents the related work that helped build and present our own proposed architecture for the dissertation planning phase.

There are two leading similar solutions to the one proposed in this document. The first is XIRAF[2] which applies automatic forensics analysis tools to evidence files from hard disk images. Once an image is inserted into the system, XIRAF executes several forensic tools from its tool repository, wraps the output, or in some cases, the Graphical user interface (GUI) and stores them in the form of Extensible Markup Language (XML) for later reporting.

The second and most recent soluction, is HANSKEN [3], developed by the Netherlands Forensic Institute, it builds on XIRAF and turns the previous stand-alone architecture into a Digital Forensics as a Service (DFaaS) [4]. Initially published in [5], this system was considered as a "game changer". It addresses challenges such as: i) Security, ii) Privacy, iii) Transparency, iv) Multi-tenancy, v) Future proof, vi) Data retention, vii) Reliability, and viii) High availability.

A common downside of these two tools is that they are not open source [6], and not easily available in the particular case of XIRAF, and in the case of HANSKEN, this is a paid service. They are also not optimized for any particular scenario as the one we consider in this work (child pornography).

2.1 Digital forensics

The advent of digital forensics has existed since computers where invented, almost forty years ago. Initially, the forensics techniques were developed mainly for data recovery. Examples of this early techniques can be found in two data recovery techniques working for 70 hours to restore a copy of a highly fragmented database mistakenly removed by a careless administrator[7].

By the mid-1980s, utility programs started being advertised that could perform tasks such as data recovery, including "Unformat, Undelete, Diagnose & Remedy"[7]. In those days, there were no automated tools for these purpose. Data forensics has emerged from a collaboration between professionals and law enforcement on an ad hoc and case-by-case basis.

Around this time, the FBI started the "Magnetic Media Program"in 1984. Unfortunately, it was only used in three cases[8].

The "Golden Age"of digital forensics were from 1999-2007, people saw it as having a magical mirror that could retrieve an artifact of the past and started to prove valuable to law enforcement agency in criminal cases where emails and instant messages were involved. At the time, it was beginning to be possible to perform network and memory recovery even months after the fact.[9]

The popularity of these techniques started spreading, also appearing in TV Series, nicknamed the "CSI EFFECT"[10].

Nowadays, Digital Foresight is facing a crisis, that is related with factors that make these activities increasingly complex and time consuming. The factors include:

- The growing size of storage devices
- Increasing embedded flash storage
- Encryption
- Popularity of cloud services
- Legal aspects related with digital evidences.

The standardization of secure encryption into operating systems has created a significant challenge for forensic examiners, meaning that it is becoming increasingly impossible to recover digital evidences without a key or passphrase[11]. This is present in hard drives and network traffic alike.

In summary, the traditional digital process are illustrated in Figure 2.1.

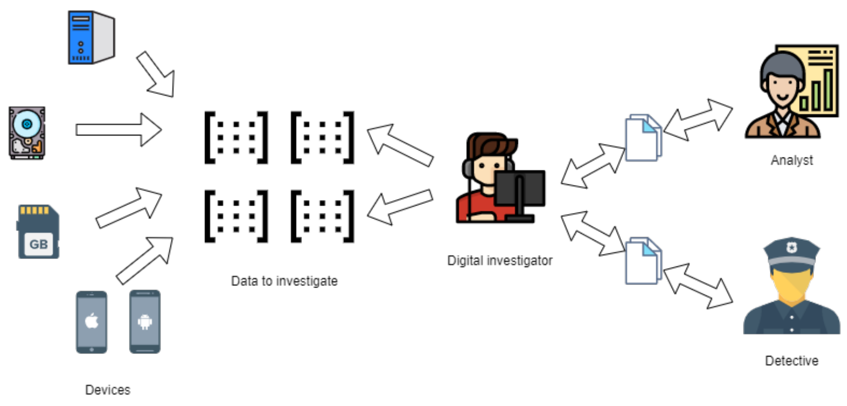

Figure 2.1: Traditional Digital Forensics process

2.2 Architecture

Since the closest work to our goal in this thesis, are the XIRAF and HANSKEN systems we will use their proposed architectures for our own soluction

The XIRAF paper [2] describes three components illustrated on (Figure 2.2), which are: the feature extraction, tool repository, and the storage system.

From the XIRAF perspective, the starting point for the operation of the system happens when one or more pieces of digital evidence are fed into the system[2]. This content is produced in the form of a Binary large object (BLOB) to be analyzed. A BLOB is a group of binary data stored as a single entity in a system capable of storing and retrieving them, these include images, audio, and other formats.

The feature extraction manager takes the input and tries to extract as many features as possible, and it achieves this by running the tools present in the tool repository and parsing output.

The tool repository contains a collection of tools for feature extraction, they are called and afterwards the outputs are merged in the form of a XML format that is stored in the storage subsystem.

The storage subsystem stores the output as an XML tree that annotates those BLOBs. BLOBs are managed by the BLOB manager who is responsible for giving access to the original BLOB input data.

Both tools and user queries require some level of access to the BLOB. To make this possible, the XML annotations are stored in MonetDB[12], a high-performance database system that provides several front-ends, including an XQuery front-end. The XIRAF architecture and modules are illustrated in Figure 2.2

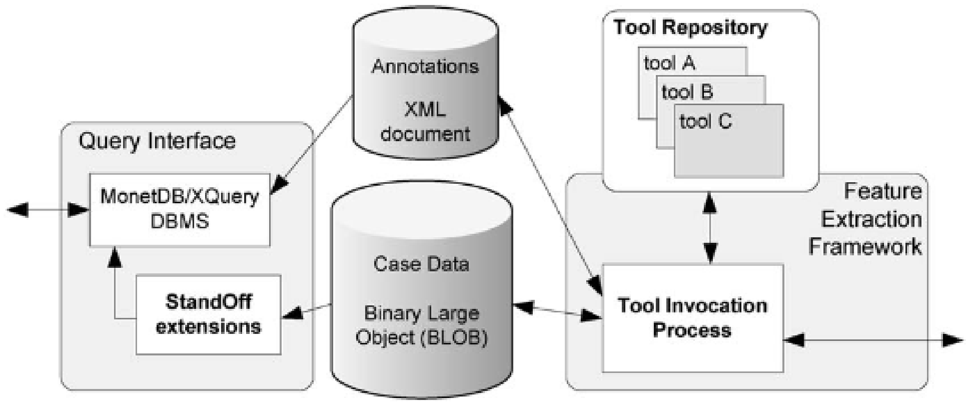

Figure 2.2: XIRAF framework architecture [2]

Looking at the HANSKEN architecture[5], we can distinguish the RPC framework, the Gatekeeper Service, the Lobby Service, the Orchestration service, the Project Service, the Keystore, the Data service, the Trace Service and a User interface.

A simplified view is provided in Figure 2.3.

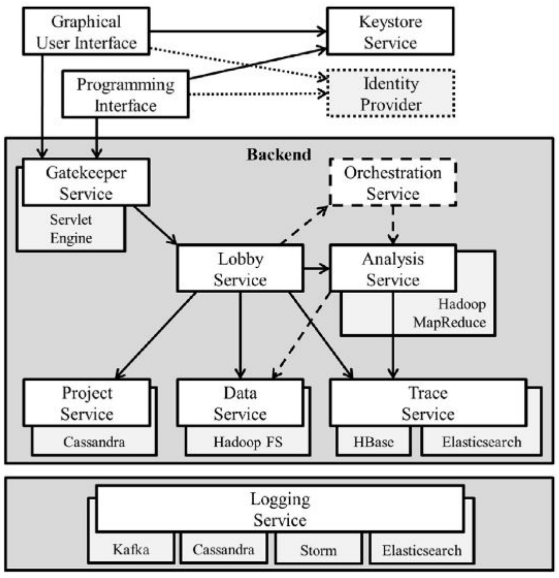

Figure 2.3: HANSKEN Modular view [5]

The RPC Frameworks is one of the most critical components of the HANSKEN architecture. It describes the communication between the different modules and the shared functionality of the modules. It includes mechanisms for initial set up and dynamic failover, where failover hosts are registered previously in Zookeeper[13].

One of the core features of this architecture is that it reports its findings to the outside world, it deals with the authentication over RESTful web service. Since our work will be executed with no external connection this module will not be described any further.

The Lobby Service funnels the user calls to their intended module, and it keeps trace of the routes that the functions should follow and assures that the requests are executed in the appropriate order.

The Orchestration Service, in theory, would make business decisions, to the best of our knowledge there is no implementation of this module, and it remains as a conceptual piece of the architecture[5]. There are however some tests using Drools[14] and the results are promising.

The Project Service is used for storing information related to images, HANSKEN agglomerates this data into cases which form a project. The unique identifier of the photos are translated into a name that makes sense to investigators. This is done on top of a key-value store. The described implementation uses Kryo[15] and stores the images and project details in the Cassandra datastore[16].

The images are retrieved from the Data Service, their implementation takes an hybrid approach model, being possible to run the Data service as a standalone, like any other module in HANSKEN, or embedded in another module for performance reasons.

The Keystore Service is responsible for storing the encrypted obfuscated keys and encrypted shared secrets[5]. The Extraction Service analyses the data and extracts from its traces, it does this by applying tools from forensics libraries and sending the results to Trace Service.

The Trace Service is responsible for storing and retrieving traces. A trace is metadata, a full keyword index, and a link to data of the trace. They used ElasticSearch[17] and stored using HBase[18].

Each trace has one or more types. These properties are generated based on another set of properties, e.g., a unique identifier, type, and name, combined with additional properties for the types of a trace, e.g., modification date, e-mail subject or phone number.

HANSKEN was designed as a service with an open interface. This allows for their clients to design and implement their own user interface. The user interfaces are not the subject of HANSKEN paper[5].

Taking into account the two previously discussed systems, there are a few components that our architecture will also need to have.

These include: a Disk image extractor, Feature Extraction, a way to store files and metadata associated with each of these files, and a GUI interface.

The Disk image extractor is needed to obtain the files present in a raw image obtained from the devices of suspects. Since there are multiple file systems, we will need a component that can operate on top of any particular file system.

The Feature Extraction is present in both architectures. On XIRAF, it is the tool repository, and on Hansken the Extraction Service, the core ideas are simple. An automated component that can run some type of metadata extractor from the files that are present in the disk image.

A crucial component is needed to store and access individual files are stored, to provide privacy and security it makes sense to have a key-value store like HANSKEN and to make it searchable potentially using ElasticSearch.

The GUI is not given any details in any of the papers, and only some screenshots are showed. In our work, we will use React because it shows code stability, ease of use, it is easy to learn. Figure 2.4 summarizes our initial proposal.

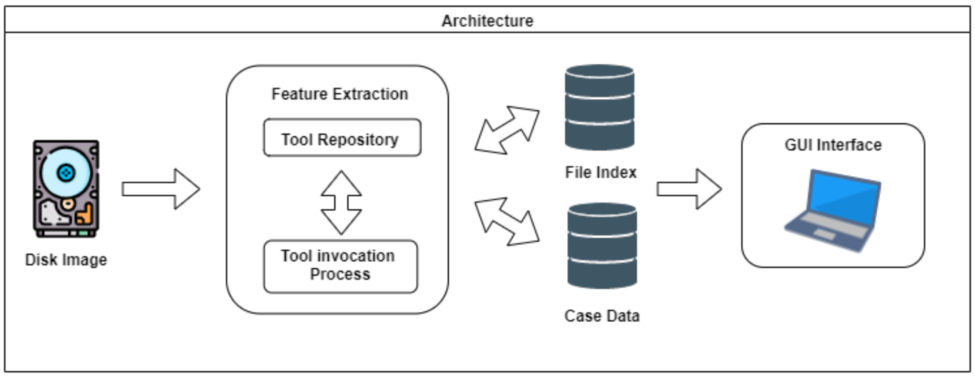

Figure 2.4: Simplified initial Architecture

2.2.1 Memory

The RAM holds the programs that the computer is currently executing. Furthermore, it is common for a program and data manipulated by other programs to be unencrypted in contrast to when they are in the hard drive, this gives forensic investigators a unique insight into what the suspect was working on, the system past state and even recover encryption keys for the content of the disk or the Hypertext Transfer Protocol (HTTP) traffic[19]. There are even several frameworks capable of automatically reassembling and redrawing all apps GUIs from a multitude of data elements present in an Android memory image. One example is GUITAR[20] which can generate 80 - 95 % accurate screenshots of what the GUI had.

Moreover, the data in memory is volatile, meaning it disappears when the power is switched off looking deeper into how data is stored in memory, it can be described as a linear supply of pages, addressed as an offset from the first page and commonly written in hex-format[19].

When the operating system needs, it can swap the memory page and store it on disk. This is done when memory is in low quantity, and another process needs additional memory space. This program is called paging. On windows, they are stored in pagefile.sys, making it a forensics artifact of value because it will survive a reboot[19].

Currently, modern computers use a technique called virtual memory, where a process is given its own virtual address space. Allowing the process to allocate more memory than its physically available and continually allocate memory[19].

In the scope of this work we do not specifically address memory forensics, with our focus being instead on hard drives and image processing.

2.2.2 Disk

Disk images are bit-streams that were extracted from physical media. They play an essential part in any forensic investigation. They can be extracted from computers and mobile devices. Nowadays they are widely used by digital forensics investigators for preserving activities, maintaining data integrity and chain of custody[21] while enabling access to potentially valuable data.

An image is ultimately a sector-by-sector copy of the data. They can also be named "snapshot" and include all allocated files, file names, and other metadata information associated with the disk volume.

Upon creation, it is stored as a single file or set of files depending on the software that was used, the simple act of turning on the device or booting the operating system can result in data being changed and possibly destroy some files. This includes modifications to operational metadata and other aspects of the original data objects such as byte order, character encoding, file system information, MAC (modified, accessed, created/changed), permissions, and file sizes[21]. Due to this, programs that manipulate the image use low-level input-output operations and without any intervention of the host operating systems. Nowadays there are available both free and commercial solutions such as FTK Imager or by the Macintosh Disk Utility.

2.2.3 Discussion

Excluding mobile drives and portable disks, the two central locations on a computer that contain data are the memory and the hard drive.

Regarding memory, it is not common for forensics investigators to obtain a memory dumps from computers, it is mostly applied to smartphones. Although there exist some forensics methods for iPhone's, there are more for Android. This is due to the fact the source code is widely available, and some examples have been discussed on [20], that show that it is possible to piece together Apps GUIs and VCR. The work presented in [22] proposes a framework capable of automatically recovering pieces of photographic evidence produced by applications.

Our work will only focus on the content found in hard disks. However, as we stated earlier, they may be encrypted [11]. There are several ways that the encryption may be broken depending on the one used. However, in this work we do not undertake this challenge and assume that the disk images provided to our system have no encrypted content.

2.3 File system

A file system is a piece of software that manages how the information is stored and retrieved. It is made of structures that make up the common content of a partition.

A hard drive contains a partition, which houses a file system that contains the data. There are some exception to this however, they are not relevant for this work (ie: swap space)[23].

The advantage of separating the data into smaller pieces and identifying each piece uniquely with a name is that it provides isolation and identity, each data block is a named file. Since it manages these structures like a system, it is named "file system".

There exits thousands of file systems, in this work we will focus on the ones that a forensic investigator are more likely to encounter, these are: NTFS (Section 2.3.1), HFS Plus (Section 2.3.2), Ext3 (Section 2.3.3) , Ext4 (Section 2.3.4), and FUSE (Section 2.3.6).

A file system can be used in several types of media, such as hard drives, removable thumb drives, optical media, and so on. The future trend for hard drive is to use Solid State Drive (SSD). This saves energy and has faster response times. The drawback is that they are more expensive.

2.3.1 New Technologies File System

The New Technologies File System (NTFS) was developed by Microsoft and is the default file system in Windows, meaning this is the most common file system for forensics.

When developing NTFS, Microsoft designed it for reliability, security, and support for large storage devices, currently in the terabyte's zone. Scalability is provided using generic data structures that wrap around data structures with specific content. This is a scalable design because the internal structure will inevitably change during normal usage[23]. To the best of our knowledge, Microsoft never published an on-disk layout for NTFS, high-level description of components can be found, but low-level details are sparse, the best we have available is documents of other research groups that describe their perception of the on-disk structures [24].

The core concept behind NTFS is that important data is allocated to files, including the administrative data, typically hidden on other file systems. Therefore, NTFS does not have a specific layout. The entire system is considered a data area. Any sector can be allocated to a file, an exception for this is the first sectors of the volume which contains the boot sector and boot code. [23].

2.3.1.1 Master File Table

The Master File Table (MFT) holds the information about every files and directories, each file or directory must have at least one entry in this table. Each entry is 1 KB in size, the first 42 bytes have a reserved purpose and the remain are used for attributes. This structure is illustrated on Figure 2.5.

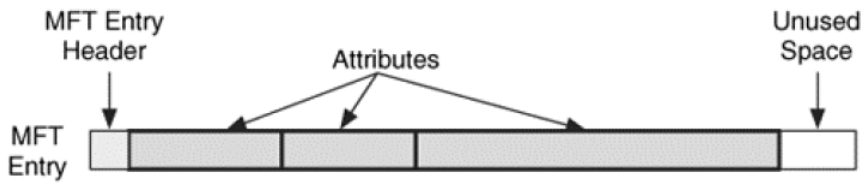

Figure 2.5: MFT entry [23]

This table is also a file and has a record to itself in the MFT which is named $MFT. The MFT location is in the boot sector, which is found in the first sector of the file system. This is illustrated on Figure 2.6.

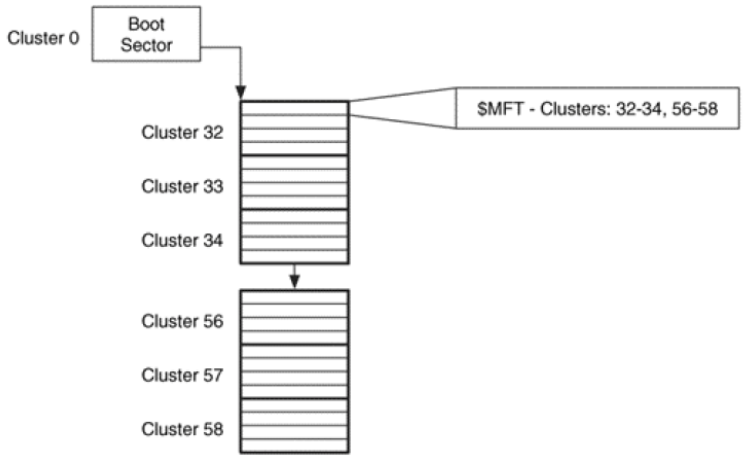

Figure 2.6: MFT Boot Sector [23]

2.3.1.2 Entry Addresses

The entries in the MFT are sequentially addressed using a 48bit value, and the first entry corresponds to value 0. Microsoft calculates the maximum MFT address by dividing the size of the $MFT by the size of each entry.

There is also present a 16-bit sequence number that is incremented when the entry is allocated, combining the entry and sequence number we obtain a 64-bit file reference address as shown in Figure 2.7.

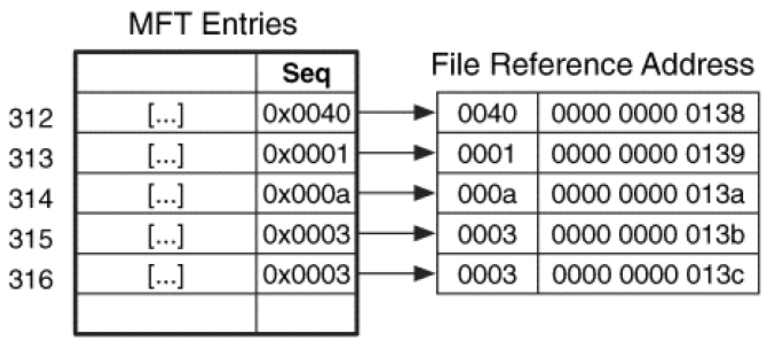

Figure 2.7: Example of the MFT entry [23]

This makes it easier to determine when a file system is in a corrupt state. An example of a scenario that can lead to this is a crash while data structures for a file are being allocated. We can use this to recover deleted content, and if we find an unallocated data structure with a file reference, we can determine if the MFT entry has been reallocated since its creation [23].

2.3.1.3 File System Metadata

Since everything in the volume has to be a file, there must exist files somewhere in the file system to store the system administrative data. These are called metadata files by Microsoft.

The first 16 MFT entries are reserved for file system metadata files, the unused entries contain only basic and generic information. These can be found in the root directory, although they are typically hidden from common users.

2.3.2 Hierarchical File System Plus

The HFS Plus is the file system used by Apple computers running Mac OS.

Initially introduced in Mac OS 8.1 [25], it shares an architecture very similar to its predecessor HFS Plus although naturally bringing several changes.

The more significant according to apple development portal [26] are summarized in the following table:

Feature	Hierarchical File System (HFS)	HFS Plus	Impact
User visible name	Mac OS Standard	Mac OS Extended	None
Number of allocation blocks	16 bits worth	32 bits worth	Decrease in space usage and files on large volumes
Long file names	31 characters	255 characters	User benefit and improved cross-platform compatibility
File name encoding	MacRoman	Unicode	Allows for international-friendly file names, including mixed script names
File/folder attributes	Support for fixed size attributes	Allows for future meta-data extensions	Future system can use metadata to improve Finder experience
catalog node size	512 bytes	4 KB	Maintains efficiency in the face of the other changes
Maximum file size	2^{31} bytes	2^{63} bytes	User benefit

Table 2.1: HFS Plus new features [26]

These features are available through the Operating system (OS) interface, since Mac OS 9.0. The key advantages they bring are an efficient use of disk space, international-friendly names, future support for named forks (see Section 2.3.2.3), and support for booting non-Mac OS.

2.3.2.1 Use of Disk Space

HFS divides the space on the volume in equal-sized pieces called allocation blocks. It uses 16-bit fields to identify a particular allocation block [26].

The use of 32-bits values for the allocation of blocks allows for 2^{32} (4,294,967,296) allocations within a volume. This means that the file system allocates more blocks resulting in a smaller allocation block size, which means less average wasted space. It also allows the file system to support more files. These changes are particularly important in volumes containing small files.

2.3.2.2 Core Concepts

HFS Plus can best be understood by discussing the core structures that manage and organize the data on the volume. These are the volume header (Section 2.3.2.4), the catalog file (Section 2.3.2.5), the attributes file (Section 2.3.2.6), the allocation file bitmap (Section 2.3.2.7), and the startup file (Section 2.3.2.8).

In the following sections we describe these structures, since understanding them can bring benefits to this work.

2.3.2.3 Fork (file system)

A Fork can have several meanings in computer science, when we refer to them in this work, it is in the context of file systems. In a file system a fork is additional data associated with an object within the file system itself. This enables the file system to have multiple sets of data for the content. This is not a common feature. However HFS and HFS Plus have native support for this[25, 26].

Originally it was designed to store non-compiled data that would help to render the systems GUI. This revealed to be very useful and so other usages started to appear, such as document processing, storing parts of a compiled resource separately, and so on. Since HFS Plus the file system can have an arbitrary number of forks (See Table 2.1).

2.3.2.4 Volume header

Every HFS Plus volume contains a 1024 byte header at the start of the volume. It includes information on the whole volume and the location of other structures. There is also a copy of this table stored in the last 1024 bytes at the end of the volume. It is intended exclusively for use when disk repair is needed. The first 1024 bytes and the last 512 MB are reserved[26], making the allocating block 1536 bytes in total, the HFSPlusVolume Header describes it, and a summary is presented in Table 2.2.

Structure	Name	Description
UInt16	signature	This field must be kHFSPlusSigWord ('H+') or kHFSXSigWord ('HX') for an HFSX volume
UInt16	version	The options for the volume format are 4 for kHFSPlusVersion or 5 for kHFSXVersion
UInt32	attributes	This atribute are an enum of strucure that contains aditional information[27]
UInt32	lastMountedVersion	A value that uniquely identifies the last mounted volume for writing
UInt32	journalInfoBlock	The number of allocation blocks which containt the JournalInfoBlock for the current volume
UInt32	createDate	The date and time the volume was created
UInt32	modifyDate	The date and time the volume was was last modified
UInt32	backupDate	The date and time the volume was last backup
UInt32	checkedDate	The date and time when the volume was last checked for consistency
UInt32	fileCount	The total number of files on the volume
UInt32	folderCount	The total number of folders on the volume
UInt32	blockSize	The allocation block size, in bytes
UInt32	totalBlocks	The total number of allocation blocks on the disk
UInt32	freeBlocks	The total number of unused allocation blocks on the disk
UInt32	nextAllocation	Start of next allocation search. The nextAllocation field is used by Mac OS as a hint for where to start searching for free allocation blocks when allocating space for a file
UInt32	rsrcClumpSize	The default clump size for resource forks, in bytes. This is a hint to the implementation as to the size by which a growing file should be extended.
UInt32	dataClumpSize	The default clump size for data forks, in bytes. This is a hint to the implementation as to the size by which a growing file should be extended.
HFSCatalogNodeID	nextCatalogID	The next unused catalog ID
UInt32	writeCount	This field is incremented every time a volume is mounted. This allows an implementation to keep the volume mounted even when the media is ejected.
UInt64	encodingsBitmap	This field keeps track of the text encodings used in the file and folder names on the volum
UInt32	finderInfo[8]	This array of 32-bit items contains information used by the Mac OS Finder, and the system software boot process.

Table 2.2: HFS Plus Volume Header field and there meaning [26]

There are several key points to take away from the previous table, the first regarding hidden files, is that we can detect their presence, chronology place the volume, and determine if it is often, or rarely accessed. Additionally, metadata can be extracted from the finderInfo array. This information can provide valuable insights, such as which files the suspect frequently open, which operations are more frequent possibility flag them for manual inspection in a later stage of the investigation.

2.3.2.5 Catalog file

HFS Plus uses a catalog file to store information about the hierarchy of the files and folders on the volume. This catalog file uses a B-Tree based implementation.

A B-tree consists of a header node, index nodes, leaf nodes, and map nodes, the location of the header node is obtained from the catalog file's header node. From this node, the operating system can search the tree for keys.

Every file or folder is assigned a unique catalog node ID know as CNID. Typically the CNIDs are sequentially allocated, starting at kHFSFirstUserCatalogNodeID, more recent implementation now allow for CNID values to wrap around enabling reuse, this is were the kHFSCatalogNodeIDsReusedBit is used to set nextCatalogID.

Since the catalog file extends a B-tree[28] file implementation it inherits the basic structures and definitions, it only changes two things: the format of the key used for the index and leaf nodes and the format of the leaf node data records.

2.3.2.6 Attributes file

The HFS Plus implementation reserves named forks for use in the future, an attribute file, is also a B-tree[28]. An attribute file is a particular file present in the HFSPlusForkData records the volume header, with no record in the catalog file, with variable length and three data record types.

A volume can have no attributes files, if it's extended attribute files contains zero allocation blocks[26]. The leaf nodes of an attribute contain the attributes, they can be Fork data which are used for attributes with large data or extension attribute which augments fork data attribute allowing the fork to have more than eight extents.

These records are known as recordType field, which describes the types of attributes that are present in the data record, the values can be a Folder Record, File records, folder thread record or file thread record. A thread record is used for linking a B-tree file to a CNID.

2.3.2.7 Allocation file

The allocation file is used to track whether each allocation block in a volume is currently allocated to a structure or not. The implementation is a bitmap. This bitmap contains one bit for each allocation block present in the volume. In the advent of a bit being set, the corresponding allocated blocks are being used somewhere in the file system structure. The opposite means it's available for allocation.

HFS provides several advantages mainly a simplified design, making it extendable and the possibility of being shrunk. The end result is an implementation that quickly creates a disk image suitable for volumes of varying sizes.

A 32-bit number determines the size of allocation blocks, the size of the allocation file can be up to 512 MB in size, HFS's only supported 8 KB.

2.3.2.8 Startup file

The startup file is specifically designed to hold the information needed to boot a device that does not have HFS Plus built-in ROM support. The boot loader can find this file without knowing the volume status (B-trees, catalog file, and so on). Instead, the volume header contains the location of the first eight extents of the startup file [26].

2.3.3 Ext3 Concepts and Analysis

The Ext3 file systems were widely used in many Linux distributions before being replaced with its successor Ext4.

Ext3 added file system journaling, but the basic structure remains the same as in its predecessor Ext2. Ext family of file systems are based on the UNIX File System know as UFS. Ext removed many of the components of UFS because they were no longer needed, which made the Ext file system easier to understand and be explained[23].

In Ext, there are two primary data structures responsible for storing data in the file system, they are known as the superblock and the group descriptor. The superblock can be located at the beginning of the file system and contains the size and configuration information, it is similar to NTFS (Section 2.3.1) and FAT32 (Section 2.3.5) file systems.

As previously stated, the file system can be divided into block groups, each of which contains a group descriptor data structure that describes the layout of the group. These group descriptors are located in a table called the descriptor table, located after the superblock. In the advent of the primary copies of the superblock being damaged more copies can be found throughout the file system in pre-determined locations.

2.3.3.1 Superblock

The Ext superblock is located 1024 bytes from the starting point of the file system and occupies 1024 bytes. In normal conditions most of these bytes are not used[23]. This structure is only used for configuration values and contains no boot code. Typically the first blocks of each group contains a copy of the superblock.

The information contained is quite simple, such as the block size, the total number of blocks, the number of blocks per block group, the number of reserved blocks before the first block group. In some instances, it also contains the total number of inodes per block group, and some nonessential data such as the volume name, last write time, last mount time, and the path in the file system where it was last mounted.

Some values can inform the operating system of the file system being clean or if some consistency check needs to be executed. The superblock can also store the total number of free inodes and blocks currently allocated.

To determine the file system layout, the first block size number is used, and the number of blocks is employed to calculate the file system size. If the result is less than the volume size, this means there could be hidden data, typically called volume slack. The first block group is located in a reserved area. Figure 2.8 illustrates this structure and also shows that the groups don't necessarily need to have the same size.

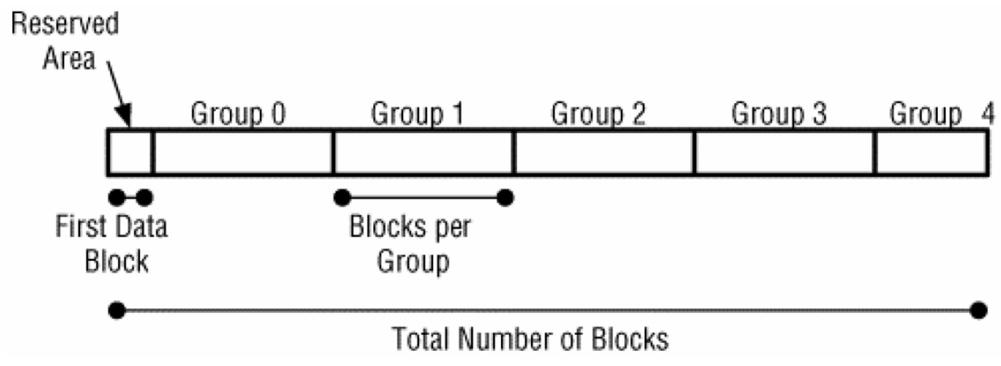

Figure 2.8: Layout of Ext [23]

In Linux, the volume label used in the superblock can identify the file system; an example of this is in Unix where /dev/hda5 refers to its device name. Another way is by using the volume label and the system configuration files. For example /etc/fstab in Linux list the file system that should be mounted and possibly refer to the /dev/hda4 device as "LABEL=rootfs" if the volume label is rootf[23].

2.3.3.2 Block Group Descriptor Tables

The group descriptor table is what comes after the superblock. It contains a group descriptor data structure for every block group. Backup of the table can exist in each block group unless the sparse superblock feature is enabled. Additionally, the block groups contain administrative data, such as superblocks, group descriptor tables, inode tables, inode bitmaps, and block bitmaps[23]. A simplified layout of the block group is depicted in Figure 2.9.

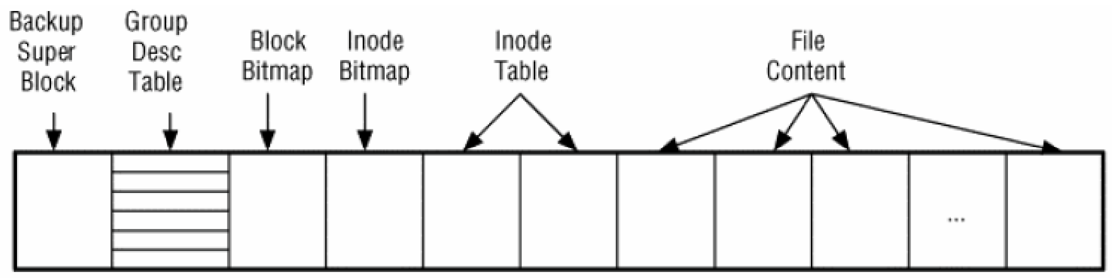

Figure 2.9: Layout of a sample block group [23]

The block bitmap manages the status of the allocation block for each group, and the starting block address is present in the group descriptor. Its size in bytes can be obtained by dividing the number of blocks in the group by eight[23]. Linux, when creating a file system, defines the number of blocks per group as being equal to the number of bits in a block. Thus, the bitmap will take precisely one bock.

The inode bitmap is responsible for keeping the state of the inodes in the group and its starting block address. Its size in bytes is the division of the number of inodes per group by eight. In practice, there are fewer inodes than blocks per group. The only exception is if the user, when creating the file system, changes these values. The starting block address for the inodes are in the group descriptor, its size is calculated by multiplying the number of inodes per group by the size of each inode, which is 128 bytes[23].

The group descriptor contains the number of free blocks and inodes in the block group. The superblock contains the total number of free blocks and inodes in all groups.

2.3.3.3 Boot Code

The boot code is not always present in every Ext file system. It only is valid if it contains an OS Kernel. All other non-boot file systems don't need this. In the cases where boot code is present, it will occupy 1024 bytes before the superblock, this means that before the first two sectors. The contents of the boot code are executed after it gains control from the boot code in the Master Boot Record (MBR) present in sector 0.

Additionally, the Ext file system knows which blocks have been allocated to the kernel and which are present in memory.

Looking at current Linux systems, many of them don't have boot code in the file system allocation. Instead, there is a boot loader in the MBR, and it knows in which blocks the kernel is located. Thus, the code in the MBR is enough to load the kernel.

2.3.4 Ext4

Ext4 is an enhancement on Ext3, currently it is the default file system for Linux distributions, this makes it an important subject for forensics investigator.

Since Ubuntu 9.10, Fedora 11, and OpenSuse 11.2, Ext4 is the primary volume file system and boot partition as well. Its predecessor, named Ext3 was one of the most highly used file systems in the Linux community as well. Still, Ext3 suffered several limitations that Ext4 addressed, the main one being the ever-increasing size of storage devices[29].

Since 2002, the need for Ext4 has been evident [30] with multiple proposed extensions for Ext3 to provide new features for maintenance and evolution. It is backward compatible with Ext3.

A significant difference in regard to Ext3 is on the organization of group blocks, the previous 128 MB block group limit resulted in a lower file system size, not ideal for modern drives [31]. This is a result of the fact that only a small number of group description could be placed in the span of a single block group. The soluction proposed for this problem is the meta-block group feature [30], where a single descriptor block can describe a group consisting of a series of blocks.

2.3.5 FAT Concepts and Analysis

The File Allocation Table (FAT) is probably one of the most simple file system present in conventional operating systems. FAT has used on Microsoft DOS and Windows 9x operating systems, but since then was replaced with NTFS (Section 2.3.1). It is supported by all Windows and most Unix operating systems and from a digital forensics point of view, it will be encountered by investigators for years to come. The reason for this is frequently found in the compact flashcard. These can range from digital cameras to USB thumb drives, which many times use this file system, which makes it an important file system to discuss.

One of the main reasons we stated earlier that the FAT file system is simple is due to its small number of data structures. However, this has made it necessary, over the years, to modify those data structures, the objective of providing it new features.

There are two main data structures in FAT that address multiple purposes and belong to various categories of the model. Every file and directory is allocated as a data structure, called a directory entry. This contains file name, size, the starting address of the file content, and other metadata. Files and directory content are kept in data units, which are named clusters, they can be allocated to more than one cluster. The relation between these data structures is shown in Figure 2.10

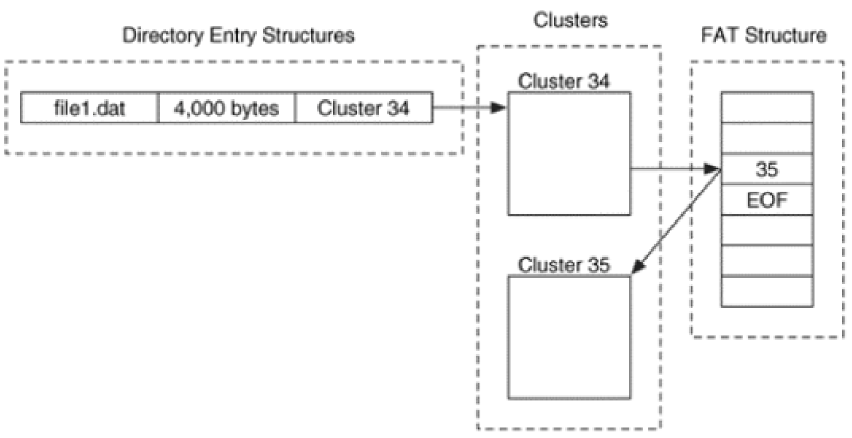

Figure 2.10: Relationship of FAT structure [23]

There are three different versions of FAT: FAT12, FAT16, and FAT32. The significant differences among these versions is the size of the entries in the FAT structure. The layout of a FAT file system is comprised of three physical sections, which can be seen in Figure 2.11.

The Reserved area is the first section and includes data in the file system category. Typically in FAT12 and FAT16, there is only one sector. It can be, however, altered in the boot sector. The second section is the FAT area, and it houses the primary and backup FAT structures. This structure is present immediately after the reserved area. The third section is named the data area and stores the cluster that will be allocated for the file or the directory.

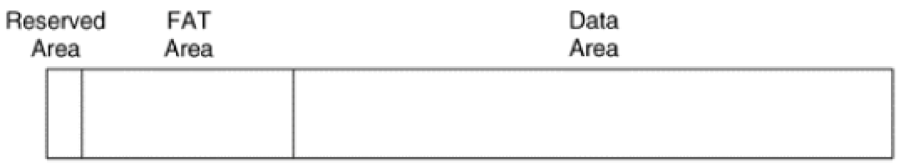

Figure 2.11: Physical layout of a FAT file system [23]

A FAT 32 file system boot sector contains additional information, that includes the sector address of a backup copy of the boot sector and a major and minor version number. FAT 32 also has a FSINFO data structure that contains information about the location of the next available cluster and the total amount of free clusters [32]. This data is not guaranteed to be accurate and is only a guide for the operating system. The next subsection will describe these aspects in more detail.

2.3.5.1 Essential Boot Sector Data

One of the first concepts that is relevant to understand the FAT file system is the location of the three previously discussed layout areas.

The data areas in FAT are organized into clusters and described in the boot sector. There are slight differences in the data area from FAT12 and FAT16 to FAT32. In FAT12 and FAT16, the beginning of the data area is solely used for the root directory, whereas, in FAT32, the root directory can be anywhere in the data area. However, it is rare for it not to be in the beginning similar to previous versions of FAT. Dynamic size and location of the root directory allows FAT32 to avoid bad sectors and enable the directory to grow as large as needed. In FAT12/16, the directory has fixed sizes that come from the boot sector. Figure 2.12 shows a detailed comparison of FAT12/16 and the FAT32 file system.

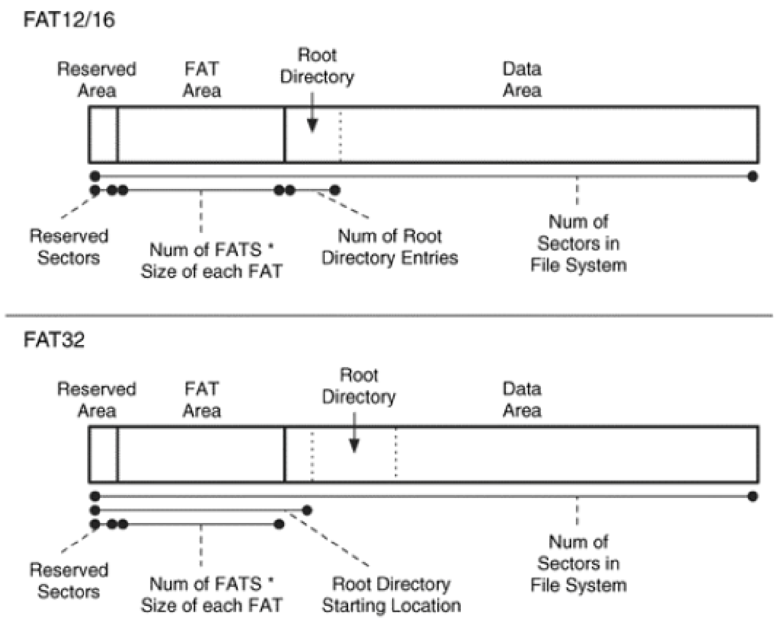
FAT12/16
Reserved Area
FAT Area
Root Directory
Data Area
Reserved Sectors
Num of FATS * Size of each FAT
Num of Root Directory Entries
Num of Sectors in File System
FAT32
Reserved Area
FAT Area
Root Directory
Data Area
Reserved Sectors
Num of FATS * Size of each FAT
Root Directory Starting Location
Num of Sectors in File System

Figure 2.12: FAT file system layout [23]

2.3.5.2 Non-essential Boot Sector Data

In addition to the layout information, the boot sector contains non-essential values. These non-essential values have these names because they are not needed for the file system to work normally. The purpose of these values is for convenience and, in some cases, may not even be correct. An example of such a value is a string called the OEM name. This corresponds to the tool that was used to create the file system. However, it's an optional value. Windows 95, as an example, sets it to "MSWIN4.0", windows 98 to "MSWIN4.1", Windows XP to "MSWIN5.0“ and so own. Linux is also capable of using FAT, it set this value to "mkdosfs"[23].

FAT file systems have a 4-byte volume serial number that according to Microsoft specifications, is generated at creation time using the current time, although the operating system can decide any value it sees fit.

Additionally, there is an eight-character string, which can be "FAT12", "FAT16", "FAT32", or "FAT". Almost every toolset sets a value for this string currently, but it is not a requirement for it to be updated. The only accurate way of determining the file system type is by calculation.

The last label is an eleven-character volume label string that can be specified by the user on creating the file system. It is saved in the root directory of the file system.

2.3.5.3 Boot Code

The boot code for a FAT file system is intertwined in the system data. This is the complete opposite of Unix file systems where it is completely isolated.

The first three bytes of the boot are jump instructions in assembly that make the CPU jump the configuration data to the rest of the boot code.

Having boot code in a FAT file system does not guarantee however, that the drive is bootable. In these cases, the code display a message to signal another disk is needed to boot the system. The boot code is called from the instructions present in the MBR, and this locates and loads the appropriate operating system file.

2.3.5.4 Content Category

The content category is where the data that comprises a file or directory is stored. Fat names its data units as clusters. A cluster is a group of consecutive sectors. These must be of the power of 2, such as 1, 2, 4, 8, 16, 32 or 64. According to Microsoft official specification[33], the maximum cluster size is 32KB. Every cluster has an address, and the address of the first cluster is 2. Meaning it's impossible for a cluster address to be 0 or 1. Every cluster is located in the data area region of the file system, which is the last of the three areas[23].

2.3.6 File system in Userspace

FUSE is the most widely used userspace file system implementation[34]. Although many file systems were implemented using FUSE because of its simple API not many work due to its internal components such as architecture, implementation, and also due to performance issues[35].

FUSE consists of a part kernel and part user-level. The kernel part is implemented in the Linux kernel module, that when loaded, registers a fuse file system. A fuse file system behaves like a proxy for the user-level daemon.

A FUSE driver is registered with the path /dev/fuse block devices, and this device is the interface between daemons and the kernel. Generally, the daemon reads FUSE requests that arrive at /dev/fuse, processes them and afterward writes the response to the same path.

A simplified and modular view of FUSE architecture is illustrated in Figure 2.13.

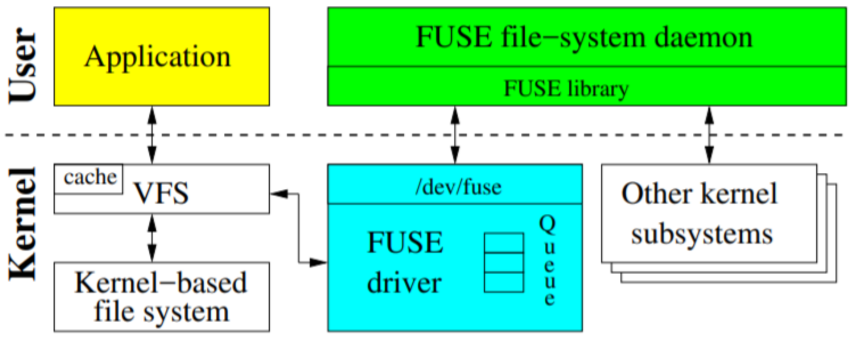

Figure 2.13: FUSE high-level architecture [36]

When an operation occurs over a mounted FUSE file system, the VFS is responsible for routing the transaction to the FUSE's kernel driver. This driver then allocates a structure for the request and places it in the queue. The process that started the operation is placed in a waiting state. Sometimes processing might require FUSE to reenter the kernel, which can generate a lot of overhead, which in turn makes FUSE slow. Recent work proposed a soluction to make a FUSE implementation work without crossing the userspace [37]. Without this, previous implementations could also complete requests without communicating with the user-level FUSE daemon, but they had to be present in the kernel page cache.

2.3.7 Discussion

After presenting a brief overview of the various file system one thing is very clear, we need a tool capable of abstracting the file system and presenting the files in a uniform manner capable of being fed in a pipeline efficiently.

The only tool capable of providing such level of abstraction is Sleuth Kit (TSK) and the Autopsy Forensic they are both Unix-based tools released in early 2001 [23].

The Sleuth Kit is a bundle of over 20 command-line tools that are organized into groups[38]. Certain groups include volume tools, disk tools, file system tools, and searching tools. Additionally, the file system tools, contains disk drive utilities, and drive volume utilities, that are capable of organizing the data into categories. Each of the tools can be broken into two parts, the first uniquely identifies its group, and the second identifies its function. An example is a file name category (starts with f) that lists (the ls) another example would be *istat* which manipulates the metadata category (the i) that displays statistics (the stat).

2.4 Machine Learning Techniques

In order to solve a problem on a computer, we need an algorithm. An algorithm is a simple sequence of instructions that operates over some input and generates some output[39].

Counterpointing this is the case were we can not create or do not have an algorithm to perform some complex task. An example is distinguishing a spam email from a legitimate one, because this requires answering the question: is something spam? and what should be taken into account?

Machine Learning tries to solve a problem where we lack in depth knowledge necessary to build a clear algorithm. Taking the example above we can quickly compile hundreds of spamming emails and try to make the computer "learn"from that data.

The advantage nowadays is that computer hardware has the ability to store and process large amounts of data.

Machine learning can be defined more formally has programming computers to optimize a performance criteria using example data or past experience[39]. This work will use models that had parameters previously defined, and they were built by infering them from training data or past experiences. The backbone of building mathematical models is the theory of statistics because it makes inferences from samples. Machine learning can be split into four basic kinds of problems, these are:

- Unsupervised learning
- Supervised learning
- Reinforcement learning
- Semi-supervised learning

Unsupervised learning, is useful for finding unknown patterns in data sets without classifying previously, it can also go by the name of self-organization because of this. It models probability densities of given inputs [40].

Supervised learning is a function that maps input to an output based on example, input-output pairs[41]. Inferring a function from labeled training data from a set of examples[42].

Reinforcement learning, sometimes abbreviated to RL, is related to how an agent should take certain actions when placed in an environment in order to maximize some notion of rewards[43].

Finally, Semi-supervised learning combines some of the approaches explained earlier. It joins some labeled data with a large amount of unlabeled when the model is in training.

2.4.1 Not Safe For Work

NSFW or Not Safe For Work are images that contain nudity and pornographic material. The detection of such material and the disciple of automatically identify corresponding imagery is also referred to has NSFW[44].

In recent years the reach in the field of NSFW detection has increased, this is due to governmental guidelines in some countries trying to limit the availability and distribution of such material. Also, companies denying access in their infrastructures to such materials requires automatic mechanisms for classification.

Some earlier approaches were based on skin detection and human body part detection using classifiers with hand-crafted feature[45, 46].

We found some implementations of this method using JavaScript and Python(nude.py).

Other techniques include using visual color words for detecting child pornography [47]. This approach aimed to help forensic investigator identify illicit images from a large image set.

However, looking at more recent methods, we see a trend in using Convolutional neural network (CNN), which have the advantage of not requiring any feature before training. CNN's can have learning algorithms that are supervised, semi-supervised or unsupervised (see Section 2.4).

The primary difficulty in NSFW is distinguishing between beach photos of people in bikinis and trunks and real pornographic content. CNN overcame this by training using large datasets which allow the algorithm to take away specific features of pornographic content has if it was a human training.

2.4.2 Age Group Recognition

Age Group Recognition is the ability of an algorithm to classify faces into predefined age groups with an acceptable accuracy[48]. It is not a simple process even for humans[49], the Mean Absolute Error (MAE) of human age estimations from visual appearance is 4.7 year[50] .

Automatic age group classification can be made more accurate by using large databases such as identification systems. Some other major fields that take advantage of age group recognition are monitoring and bio-metrics [51], because of this they ensure younger children have no access to prohibited internet pages, and when used in vending machines can control the access to alcohol and tobacco for underage people[52].

The use of this technology is possible because of significant facial changes, such as craniofacial growth and skin deformation[48]. The most common age groups are 0-3, 4-7, 8-13, 14-22, 23-35, 36-47, 48-59 and 60+. Looking at the age group distribution, we observed that they are not evenly distributed this is because younger ages experience greater changes [50].

The challenges faced in age group classification are low images, facial expressions, and facial poses [53], additional factors like genetics, gender, and race also causes people to age differently. External factors such as facial hair, glasses and cosmetic surgery may hide the real age of a person [54], thus throwing off algorithms.

2.4.3 Discussion

Analyzing our architecture 2.4, it is apparent we away in the tool repository to identify child pornography.

Using a real child pornography dataset would provide a reliable and accurate result. However, it would be illegal and immoral for several reasons. Thus we need a new way to solve our problem.

Glancing at the problem, we need a way first to identify pornography material and afterward separate a child from adult. The soluction for this is first to use an NTFW algorithm and secondly age classification algorithms[44].

There are several NSFW detection algorithms, the most popular are based on CNN architectures (see Section 2.4), they are Yahoo, Clarifai, nude.py and I2V [55–58]. Initial approaches were based on skin detection and human body parts, and they used handcrafted features, an implementation of this algorithm is nude.py [45, 46].

Yahoo released an open-source NSFW detection API freely available on GitHub in 2016.

Clarifai is a paid commercial soluction from Clarifai Inc, who specializes in computer vision, which relies on CNNs in age group identifying. Another free NSFW detection is Illustration2Vec abbreviated to I2V, trained initially with Anime images [59]. It distinguishes from other NSFW soluction in identifying explicit pornographic actions and specific body parts. For this, it is not a useful feature and will not be taken into account. Additionally, it is normal for algorithms that focus on pure NSFW detection to achieve much more precise results[44].

Each of them takes a threshold classifier. This was previously studied in [44]. They took a dataset of 2,000 sample images and calculated the ROC curves for all five classifiers, Yahoo, Clarifai, I2V, nude.py, and a coin toss. A ROC curve or receiver operating characteristic is a technique used for selecting classifiers based on their performance [60]. It shows how well a model can distinguish between two things in the content safe for work and not safe for work. The AUC value varies from 0.0 to 1.0. We want a soluction with the highest AUC possible. The results of their work are summarized in Table 2.3 where tpr means true positive rate and fpr false positive rate. Thus the best algorithm will have a high AUC, tpr and a low fpr.

Classifier	AUC	Threshold	tpr	fpr
Yahoo	0.975	0.384	0.928	0.076
Clarifai	0.963	0.682	0.922	0.121
I2V	0.896	0.090	0.826	0.190
nude.py	0.518	-	0.594	0.558
Coin toss	0.500	-	0.500	0.500

Table 2.3: Results of NSFW proposed classifiers[44]

Considering the most adequate threshold for each algorithm, these parameters need to be tested on a comprehensive dataset. Fortnightly [44] has already done this for us. They had limited resources and could not manually inspect millions of images. Nonetheless, they validated their results by extrapolating the measured on a simulated dataset of sufficient size. This is archived by generating a series of 100 sets, each comprising 1 million data points, each representing an image, in the 100 datasets 1000 images were NSFW, and the remaining 999000 where SFW or in statistics 0.1%. The results are demonstrated in Table 2.4, where r@k is the recall after k images are retrieved, Rank1 refers to the first relevant image. For our particular work, its crucial to determine if there is child pornography content of a given disk image or not. Ideally, independent of the number of images present it that image.

Rearrangement strategy	Rank1	r@100	r@1000	r@5000
random	1463.00	0.0000	0.0000	0.0050
binary (Yahoo)	144.39	0.0009	0.0071	0.0397
ranked (Yahoo)	8.74	0.0109	0.0977	0.3851
binary (Clarifai)	169.41	0.0006	0.0294	0.0283
ranked (Clarifai)	32.83	0.0031	0.0608	0.1386
binary (I2V)	324.65	0.0003	0.0030	0.0142
ranked (I2V)	41.79	0.0024	0.0234	0.0918

Table 2.4: Average performance over NSFW dataset

From these results, the Yahoo classifier is the better for our purpose, and we now need an algorithm for age recognition. Luckily for us, some competitions that challenge researcher to come up with solutions for this, one of theses examples is the ChaLearn LAP 2015 challenge on apparent age estimation with more than 115 registered teams.

The winner explained their model with great details and made available for any interested party to use [61]. They used CNNs with the VGG-16 architecture [62] and pre-trained on ImageNet for image classification. However, ImageNet has a limited number of age annotations. To overcome this, they crawled 0.5 million images of celebrities from IMDB and Wikipedia pages, at the time of writing this paper it is a public dataset for age prediction to date[61].

Deep EXpectation (DEX) of apparent age, starts by detecting the face in the image, and then proceed to extract the CNN predictions from an ensemble of 20 networks on the cropped face[61]. DEX, can be divided into five stages: input image, face detection, cropping the face , feature extraction, and finally prediction. Upon receiving the image, DEX detects a face using the Mathias detector [63] in order to crop the image the face needs to be aligned, the original image is rotated between -60° and 60° in five steps. The image chosen for the next step is the one with the strongest detection score. Afterward, the chosen picture is then fit in a 256×256 pixels square and analysed by the CNN. An estimated age value is then outputted.

The CNN of DEX was optimized on the crawled dataset. Being well documented, freely available, and proved in a contest, it is a perfect piece for our architecture.

2.5 Search engine indexing

The reason to index a large quantity of data is for later optimizing the speed and performance when finding relevant documents for a search query. Without the presence of one beforehand, the program would have to look up every document. This might not be seen like a problem if we have a low amount of documents to search, but issues start to arise as the number of document increases. The difference in some cases could be from some milliseconds to several hours.

Several techniques already exist, such as the Suffix tree, Inverted index, and N-gram index. These are designed taking into account the following factors:

- **Merge factors** - When data is placed in the index, how are the words added to the index, and is it possible to run operation asynchronously,
- **Storage techniques** - How is the index stored and is it possible for the data to be compressed or even filtered,
- **Index size** - How much computer storage the index requires to operate,
- **Lookup speed** - How much times does it take to find a specific word,
- **Maintenance** - How costly it is to maintain the index over time,
- **Fault tolerance** - How resilient the index is to corruption, bad data and data loss.

2.5.1 Suffix tree

Since they were introduced in [64], suffix trees have been one of the most used methods of choice for text indexing, because they are easy to implement and provide many important features.

It can be described as a compressed tree built by all files suffixes using their keys and positions in the text as their values.

They can be implemented in several ways that affect the performance. Table 2.5 shows some of these possible implementations. In this, s is be the size of the alphabet.

Name	Lookup	Insertion	Traversal
Unsorted arrays	$\mathcal{O}(s)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Bitwise sibling trees	$\mathcal{O}(logs)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Hash maps	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(s)$
Balanced search tree	$\mathcal{O}(logs)$	$\mathcal{O}(logs)$	$\mathcal{O}(1)$
Sorted arrays	$\mathcal{O}(logs)$	$\mathcal{O}(s)$	$\mathcal{O}(1)$
Hash maps and sibling lits	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Table 2.5: Implementation of a Suffix tree

2.5.2 Inverted index

An inverted index is similar to a database that stores the mapping from content to files.

Their primary focus is supporting fast searching and the main drawback of using an inverted index is the additional processing that is required upon insertion of a new document to the database[65].

The internal structure contains a *postings list* for every file present in the index. This postings list contains for each file a unique document identifier know as dockIds, which for increased retrieval speed are typically stored incrementally. These list can quickly be accessed by consulting the *lexicon*, which is a dictionary that serves as a lookup table for each term that exits in the list and points directly to it[66]. It's prevalent among search engines, and several architectures use it, such as ADABAS, DATACOM, and Model 204 [67–69].

2.5.3 N-gram index

N-Grams or n-word phrases use a tokenized index. The text is broken down word by word when they encounter one of a list of specified characters, then an n-gram of each word is emitted, this corresponds to the specified length.

The implementations are similar to a sliding window that moves over the word, this, in turn, produces a continuous sequence of characters that are in a specified length[70].

2.5.4 Image indexing

The previously explained indexes focus heavily on text indexing. However, some of these can also be applied to images. This is the case of inverted indexes, the idea behind it is to add an additional field to store the image. In Elasticsearch, this is known as the field key and is used for Binary datatype in the form of Base64 encoded string.

A Base64 schema is a group of encoders that convert binary to text using ASCII characters, each digit represents 6 bits of data.

An additional field like the properties mapping can serve to store the location on the image that the picture was extracted from and other metadata that we find relevant during the development phase.

2.5.5 Discussion

We need software that is optimized for distributed document indexing. A viable soluction is ElasticSearch. ElasticSearch is a document-oriented database that was build with speed and performance in mind, it is specially designed to store, manage and straightforwardly retrieve documents.

On storing documents, ElasticSearch converts it into JSON format making it queryable and retrievable. Additionally its schema-less, means that we do not need to provide a prior scheme before using it, indexes are automatically generated and fine tuned for type guessing and high precision. We can access them through its REST API.

Under the hood Elasticsearch is a collection of clusters, these are servers that save and give federated indexing and search capabilities. Behind the implementation there is an inverted index implementation (Section 2.5.2), this is also why it takes some time for a file to become searchable.

Its REST API is very well documented, and examples are abundant on the internet, this is an additional reason for why we chose it.

Chapter 3

Planning

If I had eight hours to chop down a tree, I'd spend the first six of them sharpening my axe.

Abraham Lincoln

In this chapter, we present the main modules of the project that will be developed in the upcoming months.

The work can be divided into 7 phases, the earliest functional prototype is planned for phase four around April.

After the test phase, we have planned a real-world scenario in collaboration with a local law enforcement agency. This will conclude the development cycle and lead to the conclusion phase.

3.1 Overview

In this document, we describe the architecture of automated extraction tools for forensics investigation that focuses heavily on child pornography content. However, our architecture is extensible, and it can contain more tools in the tools repository, the work presented in [2] takes the same approach. An example would be extraction from documents such as PDF, Word, PowerPoint, and so on.

Our main objective is to build a proof of concept to demonstrate that global forensics investigators can have access to new tools to speed up their work.

The system starts its analyses when a forensics investigator inputs a disk image, after feeding it to the docker images the Sleuth kit looks for files both hidden and visible and feeds them to the Feature Extraction Module.

The Feature Extraction Module creates a pipeline and determines if the file is an image or not. In case it is not an image, it is logged in the case data. In the advent that a picture is present, the tool repository is invoked, and the Yahoo NSFW and DEX of apparent age produce a combined score for the probability of the file containing child pornography.

The results are them stored by Elasticserach and made available throw an API. The front-end will be made using React.

After compiling the information gathered from the previous chapters, we can see a completed and detailed look of our planned solution in Figure 3.1.

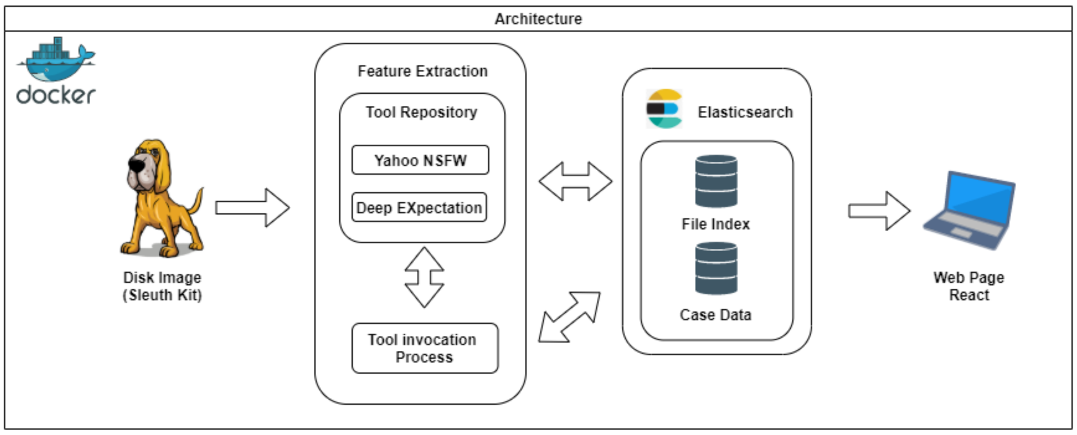

Figure 3.1: Detailed Architecture

3.2 Main task

The main task is to deliver a solution capable of processing large disk images in a reasonable time frame with acceptable accuracy.

It is not enough to identify suspicious files, the solution must also provide tags for the investigator to search for and make them admissible in court.

Additionally, it must be simple to use and require little to no technical computer background to run and maintain.

3.3 Design principles

Although our solution was never intended to be put in production in its current scale, we still hope to provide help to local and global law enforcement agencies.

Additionally, our soluction will also provide access to the seized material via a centralized service that the investigation team can depend on and rely upon. They do not need to know the back-end details of how we implemented it.

3.4 Calendarization

The project can be divided into 7 phases, which are as follows:

- **Phase 1** - Problem definition and objective

  Clear definition of the problem in collaboration with the company as well as the objectives and understatement of what the project can be.

- **Phase 2** - Related Work-Study

  Analyze the existing work, similar tools, and implementation for the different components that will be used later. The goal of this is to understand what can be useful to help speed up development for this new solution

  Additionally, try to understand if a similar product exits, both commercial or free, and if there are any limitations.

- **Phase 3** - Functional and non-functional specification

  Analyse system requirements that the final build will need to have and a guide for the system architecture

- **Phase 4** - Implementation

  The development of a base solution was described in Phase 3 and, subsequently, the components integration.

- **Phase 5** - Testing

  The various components will be tested and validated independently.

- **Phase 6** - Real-world scenario application

  Execution of the final version in a current case with real data.

- **Phase 7** Conclusion and elaboration of a report.

Table 3.1 and in Figure 3.2 presents a summary of the planning.

Descripion of Activitys	out	nov	dez	jan	fev	abr	mai	jun	jul	ago	set
Phase 1 - Problem definition and objective	X	X									
Phase 2 - Related Work Study		X	X	X	X						
Elaboration of the Dissertation Preparation report		X	X	X	X						
Phase 3 - Functional and non-functional specification				X	X	X					
Phase 4 - Implementation						X	X	X	X		
Phase 5 - Testing								X	X		
Phase 6 - Real-world scenario application									X	X	
Phase 7 - Conclusion and elaboration of a report										X	X

Table 3.1: Chronograph

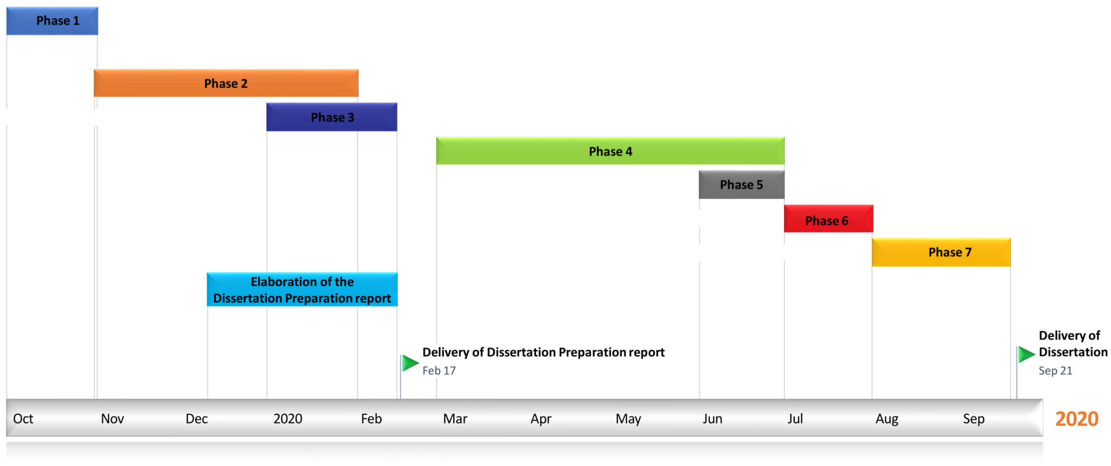

Figure 3.2: timeline of the upcoming months

3.5 Tools

Looking back at our architecture 2.4, there are three important pieces of software that are needed. Firstly we need a software capable of abstracting the different file systems and provide a common API, secondly a way to search through the documents and images that were previously found. Finally, our solution needs to be packed and made available. We chose respectively Sleuth Kit, Docker, Elasticsearch, and Docker as base technologies.

Sleuth Kit is an open-source library that contains a collection of utilities based on Unix and Windows to help forensic investigators analyze disk images. The current versions is 4.6.2 released on 9 of august of 2018 and supports NTFS, FAT/ExFAT, UFS 1/2, Ext2, Ext4, HFS, ISO 9660, and YAFFS2 file systems. Its predecessor is The Coroner's Toolkit that worked exclusively for Unix systems.

Elasticsearch is a real-time search engine that performs distributed search and analytic, and it's optimized for cloud computing environments. It is ideal for storing documents because it does not require an initial schema to be defined beforehand. Additionally, it supports structured, unstructured, and time-series queries and can be interleaved with other applications for data visualization such as Kibana. For this work, Elasticsearch is ideal because of its features concerning searching large quantities of files in real-time, other features, mainly data visualization, are not relevant. Thus, it will not be used. Future works may find some usage for these features.

Docker is an open-source tool that combines many technologies that arose from operating system research such as LXC containers, virtualization, and hash-based or git-like versioning [71]. A docker approach is similar to using a virtual machine. A key difference between using Docker and other virtual solutions relies on Docker sharing the Linux kernel with its host machine. This forces the end-user to have a Linux system or Linux-compatible software such as R, Python, Matlab, among others.

The advantage of sharing the Linux kernel is that it makes Docker more light-weight and capable of achieving greater performance. A simple way of exemplifying this is that a common computer could barely run more than two virtual machines but, in contrast, could easily run 100 containers[71]. In fact, this is the reason that made Docker so used in the industry and extremely popular. For this work, its reproducibility and ease of us is the reason we will rely on it.

The final deliverable of the thesis work will be a binary Docker image that contains all the software discussed and already installed, configured and tested.

3.6 Evaluation method

Initially, the different component will be tested using benchmark placed inside the disk images.

The YFCC100M-HNfc6 dataset[72] consists of visual features extracted from the Yahoo Flickr Creative Commons 100 Million, created in 2014 by Yahoo it contains 99.2 million photos and 0.8 million videos that were uploaded to Flickr between 2004 and 2014 made public under a Creative Commons commercial or noncommercial license. Each file has metadata associated and was created using the Caffe [73] framework.

In a later stage of development PDMFC will generate some base images that will test the output of our architecture, these will be generated from clothing catalogs which contain different age groups, gender, and races.

Finally, if the previous stage is successful, the program build will be sent to Portuguese Judicial Police to be executed in a real-world scenario and elaborate an accuracy report based on their findings.

Bibliography

[1] *Becoming A Data-Driven CEO*. URL: https://www.domo.com/solution/data-never-sleeps-6.

[2] W. Alink, R. Bhoedjang, P. Boncz, and A. de Vries. "XIRAF – XML-based indexing and querying for digital forensics." In: *Digital Investigation* 3 (2006). The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06), pp. 50 –58. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2006.06.016. URL: http://www.sciencedirect.com/science/article/pii/S1742287606000776.

[3] M. van Justitie en Veiligheid. *Hansken*. May 2017. URL: https://www.forensicinstitute.nl/products-and-services/forensic-products/hansken.

[4] R. van Baar, H. van Beek, and E. van Eijk. "Digital Forensics as a Service: A game changer." In: *Digital Investigation* 11 (2014). Proceedings of the First Annual DFRWS Europe, S54 –S62. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2014.03.007. URL: http://www.sciencedirect.com/science/article/pii/S1742287614000127.

[5] H. van Beek, E. van Eijk, R. van Baar, M. Ugen, J. Bodde, and A. Siemelink. "Digital forensics as a service: Game on." In: *Digital Investigation* 15 (2015). Special Issue: Big Data and Intelligent Data Analysis, pp. 20 –38. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2015.07.004. URL: http://www.sciencedirect.com/science/article/pii/S1742287615000857.

[6] B. Carrier. *Open Source Digital Forensics Tools: The Legal Argument*. 2002.

[7] C. Wood. *Computer security : a comprehensive controls checklist*. New York: Wiley, 1987. ISBN: 047184795X.

[8] C. of Virginia Joint Commission on Technology and Science. *Regional Computer Forensic Laboratory (RCFL) National Program Office (NPO)*. 2004. URL: http://dls.virginia.gov/commission/pdf/FBI-RCFL.pdf.

[9] S. L. Garfinkel. "Digital forensics research: The next 10 years." In: *Digital Investigation* 7 (2010). The Proceedings of the Tenth Annual DFRWS Conference, S64 –S73. ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2010.05.009. URL: http://www.sciencedirect.com/science/article/pii/S1742287610000368.

[10] H. D. E. Shelton. *The 'CSI Effect': Does It Really Exist?* 2008. URL: https://nij.ojp.gov/topics/articles/csi-effect-does-it-really-exist.

[11] E. Casey and G. J. Stellatos. "The Impact of Full Disk Encryption on Digital Forensics." In: *SIGOPS Oper. Syst. Rev.* 42.3 (Apr. 2008), 93–98. ISSN: 0163-5980. DOI: 10.1145/1368506.1368519. URL: https://doi.org/10.1145/1368506.1368519.

[12] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. "MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine." In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. SIGMOD '06. Chicago, IL, USA: Association for Computing Machinery, 2006, 479–490. ISBN: 1595934340. DOI: 10.1145/1142473.1142527. URL: https://doi.org/10.1145/1142473.1142527.

[13] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. "ZooKeeper: Wait-Free Coordination for Internet-Scale Systems." In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'10. Boston, MA: USENIX Association, 2010, p. 11.

[14] M. Proctor. "Drools: A Rule Engine for Complex Event Processing." In: *Applications of Graph Transformations with Industrial Relevance*. Springer Berlin Heidelberg, 2012, pp. 2–2. DOI: 10.1007/978-3-642-34176-2_2. URL: https://doi.org/10.1007/978-3-642-34176-2_2.

[15] EsotericSoftware. *Kryo*. Sept. 2009. URL: https://github.com/EsotericSoftware/kryo.

[16] A. Lakshman and P. Malik. "Cassandra: A Decentralized Structured Storage System." In: *SIGOPS Oper. Syst. Rev.* 44.2 (Apr. 2010), 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: https://doi.org/10.1145/1773912.1773922.

[17] H. Akdogan. *ElasticSearch Indexing*. Packt Publishing, 2015. ISBN: 1783987022.

[18] H. Patel. "HBase: A NoSQL Database." In: May 2017. DOI: 10.13140/RG.2.2.22974.28480.

[19] J. Kävrestad. *Fundamentals of Digital Forensics: Theory, Methods, and Real-Life Applications*. Springer, 2018. ISBN: 331996318X. URL: https://www.amazon.com/Fundamentals-Digital-Forensics-Real-Life-Applications/dp/331996318X?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=331996318X.

[20] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu. "GUITAR: Piecing Together Android App GUIs from Memory Images." In: *ACM Conference on Computer and Communications Security*. Ed. by I. Ray, N. Li, and C. Kruegel. ACM, 2015, pp. 120–132. ISBN: 978-1-4503-3832-5. URL: http://dblp.uni-trier.de/db/conf/ccs/ccs2015.html#SaltaformaggioB15.

[21] K. Woods, C. A. Lee, and S. Garfinkel. “Extending Digital Repository Architectures to Support Disk Image Preservation and Access.” In: *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*. JCDL ’11. Ottawa, Ontario, Canada: Association for Computing Machinery, 2011, 57–66. ISBN: 9781450307444. DOI: 10.1145/1998076.1998088. URL: https://doi.org/10.1145/1998076.1998088.

[22] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu. “VCR: App-Agnostic Recovery of Photographic Evidence from Android Device Memory Images.” In: *CCS ’15*. 2015.

[23] B. Carrier. *File System Forensic Analysis*. Addison-Wesley Professional, Mar. 2005. ISBN: 0321268172. URL: https://www.xarg.org/ref/a/0321268172/.

[24] *Linux NTFS driver*. 2004. URL: http://linux-ntfs.sourceforge.net/.

[25] *Documentation Archive*. URL: https://developer.apple.com/library/archive/navigation/index.html#topic=TechnicalNotesÂğion=ResourceTypes.

[26] *HFS Plus Volume Format*. Mar. 2004. URL: https://developer.apple.com/library/archive/technotes/tn/tn1150.html#CoreConcepts.

[27] *HFS Plus Volume Attributes*. 2004. URL: https://developer.apple.com/library/archive/technotes/tn/tn1150.html#VolumeAttributes.

[28] D. Comer. “Ubiquitous B-Tree.” In: *ACM Comput. Surv.* 11.2 (June 1979), 121–137. ISSN: 0360-0300. DOI: 10.1145/356770.356776. URL: https://doi.org/10.1145/356770.356776.

[29] K. D. Fairbanks, C. P. Lee, and H. L. Owen. “Forensic Implications of Ext4.” In: *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. CSIIRW ’10. Oak Ridge, Tennessee, USA: Association for Computing Machinery, 2010. ISBN: 9781450300179. DOI: 10.1145/1852666.1852691. URL: https://doi.org/10.1145/1852666.1852691.

[30] T. Ts’o and S. Tweedie. “Planned Extensions to the Linux Ext2/Ext3 Filesystem.” In: Jan. 2002, pp. 235–243.

[31] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. “The new ext4 filesystem: current status and future plans.” In: *Proceedings of the Linux Symposium*. Vol. 2. 2007, pp. 21–33.

[32] F. Hafeez. “Role of File System in Operating System.” In: (June 2016). DOI: 10.13140/RG.2.1.4081.8169.

[33] *Microsoft FAT Specification*. Aug. 2005. URL: http://read.pudn.com/downloads77/ebook/294884/FAT32Spec(SDAContribution).pdf.

[34] *Filesystem in Userspace*. Feb. 2005. URL: https://sourceforge.net/p/fuse/mailman/fuse-devel/.

[35] A. Rajgarhia and A. Gehani. “Performance and Extension of User Space File Systems.” In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. SAC ’10. Sierre, Switzerland: Association for Computing Machinery, 2010, 206–213. ISBN: 9781605586397. DOI: 10.1145/1774088.1774130. URL: https://doi.org/10.1145/1774088.1774130.

[36] B. K. R. Vangoor, V. Tarasov, and E. Zadok. “To FUSE or Not to FUSE: Performance of User-Space File Systems.” In: *Proceedings of the 15th Usenix Conference on File and Storage Technologies*. FAST’17. Santa clara, CA, USA: USENIX Association, 2017, 59–72. ISBN: 9781931971362.

[37] Y. Zhu, T. Wang, K. Mohror, A. Moody, K. Sato, M. Khan, and W. Yu. “Direct-FUSE: Removing the Middleman for High-Performance FUSE File System Support.” In: *Proceedings of the 8th International Workshop on Runtime and Operating Systems for Supercomputers*. ROSS’18. Tempe, AZ, USA: Association for Computing Machinery, 2018. ISBN: 9781450358644. DOI: 10.1145/3217189.3217195. URL: https://doi.org/10.1145/3217189.3217195.

[38] B. Carrier. *The Sleuth Kit (TSK) Library User’s Guide and API Reference*. URL: http://www.sleuthkit.org/sleuthkit/docs/api-docs/4.3/index.html.

[39] E. Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, Aug. 2014. ISBN: 9780262028189. URL: https://www.xarg.org/ref/a/0262028182/.

[40] *Unsupervised Learning: Foundations of Neural Computation (Computational Neuroscience)*. A Bradford Book, June 1999. ISBN: 026258168X. URL: https://www.xarg.org/ref/a/026258168X/.

[41] S. Russell. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson, Dec. 2009. ISBN: 0136042597. URL: https://www.xarg.org/ref/a/0136042597/.

[42] M. Mohri. *Foundations of Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, Aug. 2012. ISBN: 026201825X. URL: https://www.xarg.org/ref/a/026201825X/.

[43] L. P. Kaelbling, M. L. Littman, and A. W. Moore. “Reinforcement Learning: A Survey.” In: *Journal of Artificial Intelligence Research* 4 (May 1996), pp. 237–285. DOI: 10.1613/jair.301. URL: https://doi.org/10.1613/jair.301.

[44] F. Mayer and M. Steinebach. “Forensic Image Inspection Assisted by Deep Learning.” In: *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*. ACM Press, 2017. DOI: 10.1145/3098954.3104051. URL: https://doi.org/10.1145/3098954.3104051.

[45] C. Santos, E. Souto, and E. Santos. “Nudity detection based on image zoning.” In: July 2012. DOI: 10.1109/ISSPA.2012.6310454.

[46] C. Platzer, M. Stuetz, and M. Lindorfer. “Skin Sheriff: A Machine Learning Solution for Detecting Explicit Images.” In: June 2014. DOI: 10.1145/2598918.2598920.

[47] A. Ulges and A. Stahl. “Automatic Detection of Child Pornography Using Color Visual Words.” In: *Proceedings of the 2011 IEEE International Conference on Multimedia and Expo*. ICME ’11. USA: IEEE Computer Society, 2011, 1–6. ISBN: 9781612843483. DOI: 10.1109/ICME.2011.6011977. URL: https://doi.org/10.1109/ICME.2011.6011977.

[48] L. Demajo, K. Guillaumier, and G. Azzopardi. “Age group recognition from face images using a fusion of CNN- and COSFIRE-based features.” In: Jan. 2019, pp. 1–6. DOI: 10.1145/3309772.3309784.

[49] M. Kaur, R. K. Garg, and S. Singla. “Analysis of facial soft tissue changes with aging and their effects on facial morphology: A forensic perspective.” In: *Egyptian Journal of Forensic Sciences* 5.2 (2015), pp. 46 –56. ISSN: 2090-536X. DOI: https://doi.org/10.1016/j.ejfs.2014.07.006. URL: http://www.sciencedirect.com/science/article/pii/S2090536X14000501.

[50] H. Han, C. Otto, and A. K. Jain. “Age estimation from face images: Human vs. machine performance.” In: *2013 International Conference on Biometrics (ICB)*. June 2013, pp. 1–8. DOI: 10.1109/ICB.2013.6613022.

[51] R. Angulu, J. R. Tapamo, and A. O. Adewumi. “Age estimation via face images: a survey.” In: *EURASIP Journal on Image and Video Processing* 2018.1 (June 2018). DOI: 10.1186/s13640-018-0278-6. URL: https://doi.org/10.1186/s13640-018-0278-6.

[52] G. Guo, Y. Fu, C. R. Dyer, and T. S. Huang. “Image-Based Human Age Estimation by Manifold Learning and Locally Adjusted Robust Regression.” In: *IEEE Transactions on Image Processing* 17.7 (July 2008), pp. 1178–1188. ISSN: 1941-0042. DOI: 10.1109/TIP.2008.924280.

[53] E. Eidinger, R. Enbar, and T. Hassner. “Age and Gender Estimation of Unfiltered Faces.” In: *IEEE Transactions on Information Forensics and Security* 9.12 (Dec. 2014), pp. 2170–2179. ISSN: 1556-6021. DOI: 10.1109/TIFS.2014.2359646.

[54] A. Deepa and T. Sasipraba. “Age estimation in facial images using histogram equalization.” In: *2016 Eighth International Conference on Advanced Computing (ICoAC)*. Jan. 2017, pp. 186–190. DOI: 10.1109/ICoAC.2017.7951767.

[55] Yahoo. *Yahoo Open NSFW*. Nov. 2018. URL: https://github.com/yahoo/open_nsfw.

[56] Clarifai. *Enterprise AI Powered Computer Vision Solutions*. URL: https://www.clarifai.com/.

[57] Hhatto. *Nude.py*. June 2019. URL: https://github.com/hhatto/nude.py.

[58] *NudityDetectioni2v - Algorithm by sfw*. URL: https://algorithmia.com/algorithms/sfw/NudityDetectioni2v.

[59] M. Saito and Y. Matsui. “Illustration2Vec: A Semantic Vector Representation of Illustrations.” In: *SIGGRAPH Asia 2015 Technical Briefs*. SA ’15. Kobe, Japan: Association for Computing Machinery, 2015. ISBN: 9781450339308. DOI: 10.1145/2820903.2820907. URL: https://doi.org/10.1145/2820903.2820907.

[60] T. Fawcett. “An introduction to ROC analysis.” In: *Pattern Recognition Letters* 27.8 (June 2006), pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010. URL: https://doi.org/10.1016/j.patrec.2005.10.010.

[61] R. Rothe, R. Timofte, and L. V. Gool. “Deep expectation of real and apparent age from a single image without facial landmarks.” In: *International Journal of Computer Vision (IJCV)* (July 2016).

[62] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[63] M. Mathias, R. Benenson, M. Pedersoli, and L. V. Gool. “Face Detection without Bells and Whistles.” In: *ECCV*. 2014.

[64] P. Weiner. “Linear Pattern Matching Algorithm.” In: Nov. 1973, pp. 1–11. DOI: 10.1109/SWAT.1973.13.

[65] K. C. Tan. “On Foster’s Information Storage and Retrieval Using AVL Trees.” In: *Commun. ACM* 15.9 (Sept. 1972), p. 843. ISSN: 0001-0782. DOI: 10.1145/361573.361588. URL: https://doi.org/10.1145/361573.361588.

[66] G. Lavee, R. Lempel, E. Liberty, and O. Somekh. “Inverted index compression via online document routing.” In: *Proceedings of the 20th international conference on World wide web - WWW '11*. ACM Press, 2011. DOI: 10.1145/1963405.1963475. URL: https://doi.org/10.1145/1963405.1963475.

[67] V. W. Setzer and R. Lapyda. “Design of Data Models for the ADABAS System Using the Entity-Relationship Approach.” In: *Proceedings of the Second International Conference on the Entity-Relationship Approach to Information Modeling and Analysis*. ER ’81. NLD: North-Holland Publishing Co., 1981, 319–336. ISBN: 0444867473.

[68] P. J. Pratt. *Database systems: Management and design*. Boyd & Fraser Pub. Co, 1987. ISBN: 0878352279. URL: https://www.amazon.com/Database-systems-Management-Philip-Pratt/dp/0878352279?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0878352279.

[69] D. Merrit. “Volume Testing of Statistical Database Using Model 204 DBMS.” In: *Proceedings of the 1st LBL Workshop on Statistical Database Management*. SSDBM’81. Berkeley, USA: Lawrence Berkeley Laboratory, 1981, 380–389. ISBN: 155555222X.

[70] S. Huston, A. Moffat, and W. B. Croft. “Efficient Indexing of Repeated N-Grams.” In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. WSDM ’11. Hong Kong, China: Association for Computing Machinery, 2011, 127–136. ISBN: 9781450304931. DOI: 10.1145/1935826.1935857. URL: https://doi.org/10.1145/1935826.1935857.

[71] C. Boettiger. “An introduction to Docker for reproducible research.” In: *ACM SIGOPS Operating Systems Review* 49.1 (Jan. 2015), pp. 71–79. DOI: 10.1145/2723872.2723882. URL: https://doi.org/10.1145/2723872.2723882.

[72] G. Amato, F. Falchi, C. Gennaro, and F. Rabitti. “YFCC100M-HNfc6: A Large-Scale Deep Features Benchmark for Similarity Search.” In: *Similarity Search and Applications*. Ed. by L. Amsaleg, M. E. Houle, and E. Schubert. Cham: Springer International Publishing, 2016, pp. 196–209. ISBN: 978-3-319-46759-7.

[73] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. “Caffe: Convolutional Architecture for Fast Feature Embedding.” In: *CoRR* abs/1408.5093 (2014). arXiv: 1408.5093. URL: http://arxiv.org/abs/1408.5093.