Department of Computer Science

Therese Axelsson and Daniel Melani

Hash Comparison Module for OCFA

Bachelor's Thesis

C2010:17

Hash Comparison Module for OCFA

Therese Axelsson and Daniel Melani

This thesis is submitted in partial fulfillment of the requirements for the Bachelor degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Therese Axelsson and Daniel Melani

Approved, 2010-06-09

Advisor: Johan Garcia

Examiner: Martin Blom

Abstract

Child abuse content on the Internet is today an increasing problem and difficult to deal with. The techniques used by paedophiles are getting more sophisticated which means it takes more effort of the law enforcement to locate this content.

To help solving this issue, a EU-funded project named FIVES is developing a set of tools to help investigations involving large amounts of image and video material. One of these tools aims to help identifying potentially illegal files by hash signatures derived from using classification information from another project.

Contents

List of Figures

Chapter 1

Introduction

Paedophilic content on the Internet is today a problem difficult to deal with since the files are hard to identify and locate. To deal with this issue, a EU-funded project lead by Karlstad University is developing a set of tools to decrease the amount of harmful files on the Internet.

This paper is a documentation of the development of one of these tools. The purpose for this tool is to be able to search for computer forensic evidence more effectively. It is interesting as a bachelor thesis because it involves data structures and hash algorithms, which are important parts in the computer science curriculum at Karlstad University. Since the tool was implemented in the programming language C++, this bachelor's project also provides a good opportunity to evolve in this area of computer science.

As this project was related to a number of other projects, it was also essential to be involved in how the different projects work and take this into consideration in both research, design and implementation.

No pre-defined working method has been used; the module and documentation has been developed primarily at two PC's with Ubuntu at the Carl lab at Karlstad University.

1.1 Disposition

This section provides a short presentation of the five chapters in this paper.

- Chapter 2 covers the necessary background information used during the development and implementation.
- In chapter 3 a short presentation of the preparation and design of the development is given.
- Chapter 4 explains the implementation in detail.
- Chapter 5 is a discussion of the projects result and evaluation as well as a presentation of the problems dealt with.

Chapter 2

Background

This chapter describes the background information for this paper, both from a technical and project oriented point of view. As the objective for this project is to develop a new module for an existing system, some technical knowledge is needed and the most essential details will be presented in section 2.1. This will be followed by an introduction of other projects that are related to this project.

- Section 2.2: OCFA – Open Computer Forensics Architecture
- Section 2.3: FIVES – Forensic Image and Video Examination Support
- Section 2.4: MAPAP – Measurement and Analysis of P2P activity Against Paedophilic content

Initially an introduction will be given to a computer forensic framework which consists of several programs that runs separately on top of a tailored library inside the architecture. After that, two EU-funded projects which aims to reduce the amount of child abuse content on the Internet will be described.

2.1 Technical background

This section will cover all the technical dependencies needed to reach the objective of the project. These dependencies covers some database interfaces, networks and algorithms. A discussion of differences between different technologies will as well be given.

2.1.1 eDonkey

eDonkey is an advanced file sharing program which uses peer-to-peer [21](P2P) technology. It was initially the original client for the eDonkey2000 network, also called ED2K [18]. eDonkey is well suited for large files such as software, image, audio and video files as it is adapted for this kind of use. Similar to torrent technology, the client can download small pieces of the file from many different peers and as it is P2P, the network is decentralized. An important quality of eDonkey is the functionality of searching; instead of retrieving results by comparing file names it uses the files hash [18]. A file hash is what one may call a file's fingerprint which consists of a string of characters. Two different files are nearly guaranteed not to have the exact same hash, despite how similar they are. They are therefore optimal for file searching as when two hashes are identical, they are copies of the same file [19]. The file sharing client eDonkey is no longer developed nor supported since 2005 whilst the eDonkey network is kept alive by its users.

2.1.2 Hash lists

To ensure data integrity when transferring files, a hash list can be used. A hash list is an expansion of the concept of using hash values of data to ensure its integrity. As shown in Figure 2.1, instead of using a single hash value for all the data, the data is split into several blocks of data. A hash string is then calculated separately for each block, then a top hash is calculated as a sum of all hash pieces [20].

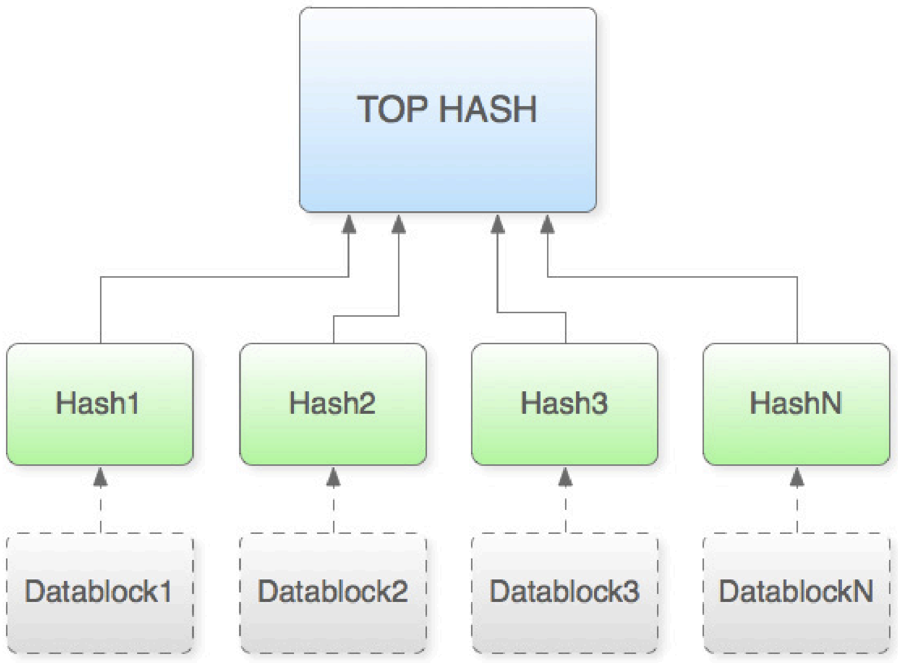
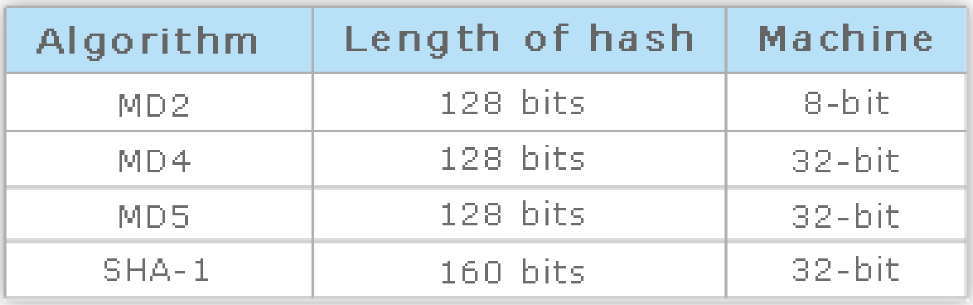


Figure 2.1: Illustration of how a hash list is built

When downloading files hash lists has an advantage over a regular hash. If there is an error present in the downloaded file and a single hash value is used, the mismatch of hashes would indicate an error and the entire file would have to be downloaded again. However, if a hash list is used, only the file segment containing the error would have to be downloaded.

2.1.3 Message Digest Algorithms

A message digest algorithm is an algorithm which produces a hash string, or message digest, from a file or text string of an undefined size. A hash string that is produced in this way is not decodable, which means it is irreversible. Commonly used message digest algorithms are MD2, MD4, MD5 and SHA-1, all suitable for different purposes. As illustrated in the table in Figure 2.2, the hash string produced by the algorithms are of different lengths and the algorithms are developed with different machine architectures in mind.



Algorithm	Length of hash	Machine
MD2	128 bits	8-bit
MD4	128 bits	32-bit
MD5	128 bits	32-bit
SHA-1	160 bits	32-bit

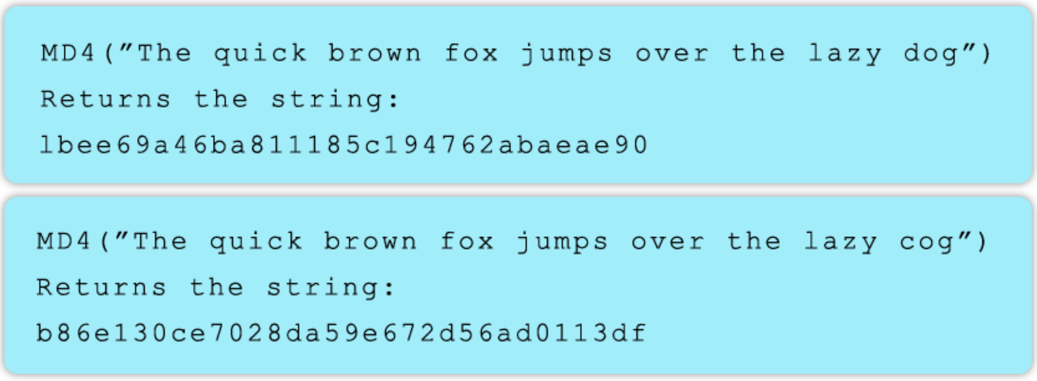
Figure 2.2: Table over different message digest algorithms

Though the hash strings have fairly similar length, the method of producing them differ [9][6].

2.1.4 MD4 – Message Digest 4

MD4 is the fourth in the message digest series and is considered fast and compact. The leap between MD4 and its predecessors is quite large as MD2 was made to compute on 8-bit machines [10]. As opposed to the relation to the predecessors, the differences between MD4 and the successor MD5 are not large. MD5 is as well developed for 32-bit machines and work similarly but with more security functionality. Thus, MD5 is essentially the same algorithm as MD4 but more secure. This algorithm was developed as a result after discovering some safety loopholes in MD4 especially on PC machines [6].

The hash string is computed by taking a message of limited undefined length and returning a hash string with a length of 128 bits expressed as a hexadecimal of 32 characters. Even a small change in the message, as small as one single character, result in a substantial change in the digest message returned, which makes it in theory infeasible for MD4 to produce two message digests with the same output value [17]. To ensure that the messages length plus 64 is divisible by 512 it is padded with the length required.



```
MD4("The quick brown fox jumps over the lazy dog")
Returns the string:
1bee69a46ba811185c194762abaeae90
```

```
MD4("The quick brown fox jumps over the lazy cog")
Returns the string:
b86e130ce7028da59e672d56ad0113df
```

Figure 2.3: Example of generated hash strings [17].

Figure 2.3 illustrates clearly that the generated hash changes dramatically when only replacing the letter d with c.

2.1.5 POSIX

POSIX is a standard for an operating system's (OS) API and other interface utilities. POSIX is short for Portable Operating System Interface and is implemented at many of the UNIX-like OS's [22].

2.1.6 PostgreSQL

PostgreSQL, developed by the PostgreSQL Global Development Group, is an object-relational database management system [7]. PostgreSQL has its origin at the University of California, Berkeley. The database management system was started as a post-Ingres project and used many ideas from an earlier system called Ingres. The project was named Postgres and was lead by Michael Stonebraker, who also led the projects predecessor Ingres [7].

In 1986, the first papers describing the basis of PostgreSQL were released, and in 1988 the first prototype was made available. The focus was to remedy a number of, at the time, common problems with database systems. The new features made it possible for a user to fully describe the relationships within the database. This was earlier something that was handled by the user and not the database.

PostgreSQL strongly adheres to the ANSI-SQL 92/93 standards with support for read-committed serializable transaction isolation levels, and full support for sub queries. PostgreSQL has a number of features and extensions that are not part of the standards. There is also support for a wide variety of data types.

Blocks of code can be organized into functions which can be executed on the PostgreSQL server. These functions are useful when a user wishes to extend PosgreSQL's features. For example, a user may wish to add functionality for different mathematical algorithms, cryptography, or even Oracle compatibility.

The set of programming languages in which functions can be written in are:

- Compiled languages – C, C++, Java
- Statistical language – R
- Scripting languages – Lua, LOLCODE, Perl, PHP, Python, Ruby, sh, tcl, Scheme
- Built-in language – pgSQL

Access to PostgreSQL can be done through library interfaces. These interfaces exist for many different programming languages. Examples of languages for which there exist a library interface are: C, C++, Java, Perl, Python and LISP. Documentation for PostgreSQL is rich and can be found on the projects web site. PostgreSQL also has a large user community from which many code examples can be found.

2.1.7 Berkeley DB

Berkeley DB was created when BSD went from version 4.3 to version 4.4 within the time period 1986 to 1994. The main reason for creating Berkeley DB was to remove dependency on source code developed by AT&T. From 1996 to 2006 the database was developed by Sleepycat Software, a company created when Netscape requested that the authors of Berkeley DB improve the application. Sleepycat software was acquired by Oracle Corporation in 2006 which is still developing the software [16].

Berkeley DB has, compared to many other database implementations, a very simple architecture. The database system is implemented as a library. User created programs access database through API calls.

In Berkeley DB there is no support for a query language, SQL or similar. There are no table schemas or table columns. How the data and keys are stored is left to be decided by the user. Advanced features such as ACID transactions, fine-grained locking, hot backups, and replication are, despite Berkeley DB's simple architecture, implemented.

Berkeley DB is supported for Windows and operating systems using the POSIX (see section 2.1.5) standard, presented in section 2.1.5 [4].

2.1.8 Differences between Berkeley DB and PostgreSQL

There are a number of important differences between the two database management systems PostgreSQL and Berkeley DB. With both database management systems there are a number of benefits and drawbacks and there is no obvious choice between PostgreSQL and Berkeley DB.

PostgreSQL is a complete object-relational database management system with a large feature set, standards compliant and equipped with many and advanced extensions, while Berkeley DB is a high performance embedded database with a simple architecture. Berkeley DB stores data arbitrarily in byte vectors and the representation of the data is left to the user, while PostgreSQL has support for many different kinds of data types and facilities for creating user defined data types. There is no query language in Berkeley DB, and there is no network support either. PostgreSQL, on the other hand, has support for both.

The two databases are distributed under different licenses. PostgreSQL is distributed under the PostgreSQL license [7]. Berkeley DB is distributed under the Sleepycat License [16].

2.2 OCFA – Open Computer Forensic Architecture

OCFA is short for Open Computer Forensics Architecture and is an Open Source computer system whose purpose is to facilitate the search of harmful or illegal files [11]. It is a framework which brings together several modules with different purposes and the environment allows existing tools and libraries to be easily plugged in.

The first PoC version was completed in the year 2004 by the Dutch National Police Agency. In 2006 the first complete live version, 2.0.0, was released and now as an Open Source software system licensed by GPL/LGPL. The current version as of this day is 2.2.0 and was released April 2nd 2009 [14]. The framework is growing rapidly since new modules from different sources are added regularly.

It is developed to support processing of terabytes of data and files where the result is potential evidence used by forensic analysts. In this way, the amount of files that has to be manually processed is substantially decreased. The data, processed or pre-processed, is easily accessible through a search and browse graphical interface [15].

This paper describes the previous OCFA release 2.1.0 while the latest release is of OCFA is 2.2.0 [11].

2.2.1 Technical introduction

The OCFA framework is built on a Linux system but can be run on any operating system using the POSIX standard. OCFA is in theory a set of modules, where every module is run as a separate program. The modules are collected in the framework to form one working system. The framework is divided into several different sections which all have a unique purpose.

The fundamental requirements of the OCFA functionalities are

- The result (metadata and evidence file) must be open to a third party tool used by investigators to retrieve.
- The framework should be easy to extend, meaning development and implementation of a new module must not be complicated.
- It must be possible to run different components of the framework on separate machines and still function as one whole framework. This is to be able to ease the workload on one single machine.
- Several investigators must be able to search evidence and metadata at the same time.

The system overall is very stable since if one single module fails, the rest of the architecture remains intact thus in the event of a module crash, the data and metadata should be recoverable.

The programming language used to develop the framework is mainly C++, and SQL for the database. OCFA is developed using object oriented programming (OOP) where every module is defined as a separate object. The framework is divided into three subprojects [14]; OcfaLib, OcfaArch and the OCFA modules.

The OFCA library holds the foundation on which the entire framework is built and the most essential libraries will be described in this thesis.

The OcfaArch can also be referred to as the OCFA core modules as they set the ground rules for the basic functionality. There are a number of modules considered to be core modules, however, a few of them set rules for the most important parts of the system, such as functionality for user interface and routing.

- The AnycastRelay
- The Router Module

- The Kickstart Module

At last, the different types of modules and storage in OCFA will be presented.

2.2.2 OcfaLib

OcfaLib is the frameworks library and is accessible and used by all modules along with the other subprojects. One of the purposes of OcfaLib is to enable different modules to communicate with each other, and therefore should be included in every module. The OcfaLib is actually a collection of many smaller libraries intended for different purposes and much of the implementation of the framework is defined there.

Only relevant libraries will be presented in this paper but further reading is available at [13].

- Misc library
- Evidence library
- Treegraph library
- Fs library
- Message library
- Store library
- Facade library

All listed libraries except for Misc is part of the higher level Module library. A short presentation of the main functionality of each library will be given below.

The OCFA misc library is a collection of classes which defines a set of basic data types that are unique for OCFA, such as types for date and time. Other functionality it provides concerns logging and exception handling. The OCFA mother class, OcfaObject, is also defined here and most modules inherit from this class.

The OCFA evidence library provides an API for other modules to extract metadata to and from an XML file to obtain evidence information and add new metadata. The metadata is divided into two different levels; top level metadata and job level metadata. The API provides for extracting both levels. The top level metadata contains fields concerning the entire evidence used to locate or identify the evidence. This can be an id (identification number), src (physical source of where the evidence is located) and the MD5 (message digest 5) hash of the file itself. The job level metadata is information about the evidence added to the metadata by other modules.

The OCFA treegraph library is the foundation of all modules using advanced treegraphs to extract evidence metadata on all levels, for example some of the dissector modules (see section 2.2.6). Any module using this library can work with any level of metadata and the derived metadata.

The OCFA fs library (named evidence tree library in later releases), file system abstraction library, is formed by the treegraph library and uses the treegraph to traverse a directory tree in order to gather data and metadata on evidence on all levels. This is done by mapping the data into a structure which is easily accessible by the modules [2].

The OCFA message library contains functionality to provide processes and other modules the possibility of sending shorter messages to each other. A message can be of the following kinds:

- Unicast – messages directed to a pre-defined instance of modules

- Broadcast – messages directed to all connected modules
- Anycast – messages directed to a specific module type in any instance
- Multicast – messages directed to all modules that are subscribing a certain channel

For sending evidence and metadata, the Anycast is used. The message server itself is named the anycast relay, read further at section 2.2.3.

The OCFA store library provides functionality for modules to store evidence data and metadata in a way to completely avoid redundancy. This is achieved by hashing the evidence data with a MD5 hashing algorithm, returning a unique string. If new evidence should be stored and an exact match of its hash string already exists in the database, those will share the same space in the database [12].

The OCFA facade library is a collection of the most frequently used libraries for the OCFA modules. It is defined as a namespace and including this gives access to the majority of what is required to develop a new module.

2.2.3 The Anycast Relay

The AnycastRelay is OCFA's messaging server. It is a robust and secure messaging service and holds functionality for recovering of lost messages in case of a module crash. It is able to do this by assigning different priorities, buffering the messages and keeping status flags. As long as a message is pending, the module stays updated with which connections are active and which messages has been sent where in order to be able to resend them.

The AnycastRelay also functions as a connector and keeps a frequently updated map over which modules that are connected to the router. Not all modules within the framework are constantly active, or even started, but are initiated when called by another module [14].

Instead of having a large number of connections active throughout the framework between separate modules, connections are passing through the AnycastRelay.

2.2.4 The Router Module

The Router module has to be running in order to enable modules to have access to the incoming evidence. As its name states, the router module's purpose is to route evidence between modules. This is achieved by sending the evidence data and metadata in means of messages. For the module to be able to complete the evidence transfer, it only needs access to the top level metadata attached to the evidence which gives some information to enable the module to make a routing decision. Any other data connected to the evidence remains untouched.

All evidence carries information about the path it has travelled within the framework and also XML code containing job statuses. The status of evidence indicates whether certain task has been completed or not. When a module receives this evidence, it checks for any job status for tasks connected to that module that are not completed. Once it has completed the task, the module sets the status label "processed" and returns the evidence back to the router. The router looks for any status set to "processed" and sets it to "done". This helps the router to make a routing decision.

To be able to locate which module that should receive the evidence next, the router uses a rulelist. The rulelist is a table which is predefined by different modules, thus when implementing a new module it has to be added to the rulelist. When the metadata of certain evidence matches a certain rule, the action set for that rule is executed by the router module to route evidence forward.

2.2.5 The Kickstart Module

The kickstart module is the frameworks evidence provider. The process of submitting the evidence into the framework is called "kickstarting", thus the name of the module. What the module does is storing a link, or path, to where the evidence is stored physically but the evidence data itself is not handled by the module.

Evidence files in OCFA are simply hard disk images from where the evidence was initially located. The link can either lead to the evidence file itself or a directory where more than one instance of evidence may be located.

2.2.6 Different OCFA Module types

A module is a program which adds some functionality to the evidence processing, such as adding or extracting evidence data or metadata. Depending on what their functionality is, they can be divided into three different categories:

- Extractor module
- Dissector module
- Expander module

Depending on which type of module, different functionality from OcfaLib is given.

Extractor modules has the capability to access the actual file data of an evidence and then depending on the modules function, doing some work on the file data. The given result is added as new metadata to the evidence.

A **dissector module** has a similar functionality with the difference that it can create new evidence file data from the evidence it works with. As opposed to an extractor module, it can work with larger evidence that is separated in a directory hierarchy. This is possible by using functionality provided by the treegraph library. An example is a module that can unzip a compressed file, thus creating new evidence from the result.

At last, an **expander module** can compare the metadata of an evidence with another source of information. If new metadata is found in that source, it can combine them and create new metadata.

2.2.7 Storage in OCFA

OCFA is currently using both PostgreSQL and Berkeley DB (see section 2.1.6 and 2.1.7) to store the evidence file and metadata and is divided into two groups of database tables; the core tables and tables storing various metadata about certain evidence. The core tables store location of evidence and metadata, and also the relations between evidence and its metadata to keep track so they do not get lost.

2.3 FIVES

FIVES, Forensic Image and Video Examination Support, is a project that gathers law enforcement, academia and industry to deal with the problems with media files containing child pornography and is planned to extend over a period of two years; February 1st 2009 to February 1st 2011 (see Figure 2.4). It is an EU-funded project and Karlstad University (KAU) has the role of coordinator.

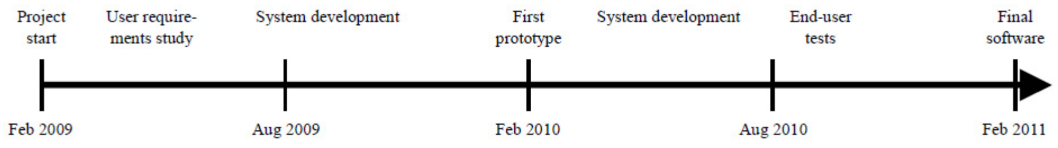


Figure 2.4: Timeline of the FIVES project [8].

To achieve the goal of decreasing the amount of child abuse content, FIVES is developing a set of tools tailored for this sole purpose. The tools are built as modules as part of the OCFA framework which in turn is set on a Linux operating system. As illustrated in Figure 2.5, the architecture includes modules for graphical user interfaces (GUI), decompressing archive files (extractors, E), five image and video processing (I and V) and last a module for specialized in hashing (H).

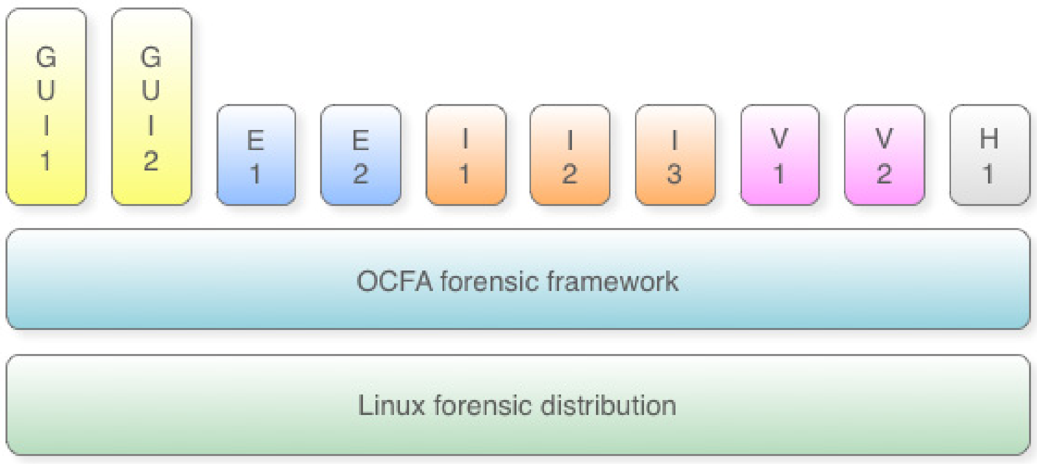


Figure 2.5: Overview of the FIVES tools architecture [8].

The FIVES modules are developed to handle media files, such as image and video and together they will generate rating for how probable it is that the file in question contains child pornography or evidence of child abuse. The functionalities to obtain this can for example be face and skin color recognition. As much paedophilic content have misleading file names and keywords, the files are usually more difficult to find. The tools developed by FIVES result in making these fake names toothless as it is the content of the file itself that is analyzed. Several graphical user interfaces can be provided for, including a windows-based front end. To enable development and testing of the tools, FIVES collects data of harmful files from the law enforcement. Further, some changes has been made to the standard OCFA functionality [8].

2.4 MAPAP

MAPAP, Measurement and Analysis of P2P activity Against Paedophile content, is a project started in year 2008 in France by partners from France, Ireland, Poland and Slovenia as a result of a french study in 2005 about the availability of paedophilic content on the Internet. The study indicated that much of this content was spread in P2P networks.

The project aims to reduce the problem of locating these harmful files as most of them use misleading file names. The law enforcements are approaching this problem by making registers over keywords often used by paedophiles to search files. Though, as the keywords are changed frequently, the register has to be updated very often[5]. A content fake rating system has been developed to counter this problem.

2.4.1 Content Rating and Fake Detection System

The content fake rating system is a tool whose future purpose is to enable Internet service providers to filter their traffic. The functionality of the fake rating system is to analyze files sent by peer-to-peer and by various measurements rating the probability of if the file name is fake, which means if the file has pornographic or paedophilic content while the file name indicates otherwise. Instead of focusing on the file name, an estimate is made by the fake rating system of how probable it is that the file has paedophilic content. This is done by examining file names, looking at the user's normal Internet habits and which files are jointly shared within user groups and similar techniques.

MAPAP has focused on the eDonkey network, a peer-to-peer network commonly used for larger files, such as image and video files. Three different techniques were used to collect data on files in the network; server-side measurement, measurement by client sending queries and measurement by honeypots. In server-side measurement, a program is recordning all queries to and from a certain server, though no information can be given if the data in question were actually received. The second technique simply means that a client send queries to a server and analyze the data that is returned and in the third and last technique, a client uses honeypots, files with names that may be appealing to paedophiles, to record where queries to get these files come from.

In this way, the MAPAP project has collected a large amount of field data stored in databases available for research use [1].

Chapter 3

Design and Components

This projects ambition was to develop a new module for the OCFA framework. The module is named MAPAP module and in this chapter a brief presentation of the design and components of the development and implementation will be given. Also a short presentation of the MAPAP module project will be covered.

Initially, the languages used for the development will be described followed by development platforms and chosen operating system. There will also be a discussion of the motivation for the languages and environments of choice.

Further, an overview of the MAPAP module project will be discussed and then the technical aspects; design and architecture.

3.1 Environment and language

This section covers which different programming languages that were used to develop the components of the MAPAP module, as well as the environments used to implement the new module.

3.1.1 Language

From the multitude of available languages to choose from, covering different paradigms and having different strengths and weaknesses, the languages chosen for the implementation of this module are C++ and Perl. The reasons for choosing these languages are many, and they will be discussed here.

For the implementation of the MAPAP module the most obvious choice of language is C++ since the entire OCFA framework is written in C++.

Perl has been chosen because it is widely available and because it has good functionality for the manipulation of text. Perl is available for most Unix system and comes pre-installed on many Linux distributions, which is one of the main reasons for being used in this project. The good support for regular expressions is also one very important feature that Perl has since Perl was used mainly to populate the database with hash values. This task was simplified by the use of regular expressions.

3.1.2 Environments

The MAPAP module was deployed in a Linux environment as it is required to run OCFA. The type of environment also affects the set of libraries and tools available to the developers. Since the given environment is Linux, there is an additional benefit which is that the module will be portable between systems, as long as the system conforms to the POSIX standard.

The reason for choosing Linux, and more specifically Debian, as the development environment is a straight forward one. The documentation of OCFA available at the time when the implementation of MAPAP was started covered the installation and configuration of Debian best.

As choice for database environment, the object-relational database management system (DBMS) PostgreSQL was used and bash scripts was used for different tasks such as populating the database. The motivation for this is that OCFA uses PostgreSQL and using the same facilitates the communication between the MAPAP module and the rest of OCFA. More of the motivation for chosing PostgreSQL will be discussed in section 3.2.

The programming environment used were Eclipse with the C++ development kit (CDK). As not all programming was performed on a Linux machine, this was a suitable choice.

3.2 Design

3.2.1 Hashing

In a peer-to-peer network, different parts of a given file may come from different sources. This presents a problem regarding the authenticity of the received data. A user may insert fallacious data into a peer-to-peer network if there is no way for a receiver to authenticate the hash list. Because each file block has a hash, malicious users can insert fallacious data into a block and calculate a new hash. The receiver will then only know if the data was transferred correctly and not the data the receiver actually wanted.

To remedy the problem of authenticity of the received data, a top hash is used. A top hash is a hash calculated based on the hash list created from all the received data blocks. To calculate the top hash, the hashes in the list are concatenated into a single sequence. A hash algorithm, not necessarily the same algorithm used for the hash list, is then applied to the sequence, and thus the top hash is generated. If the top hash is acquired from a trusted source then the authenticity of the hashes in the hash list can be verified.

3.2.2 Hashing algorithm choice

The module created in this project uses a database containing MD4 hash values. The values within the database are collected from a peer-to-peer file sharing program called eDonkey, presented in section 2.1. Therefore the module must use MD4 in the same manner as eDonkey in order to generate valid MD4 hash values.

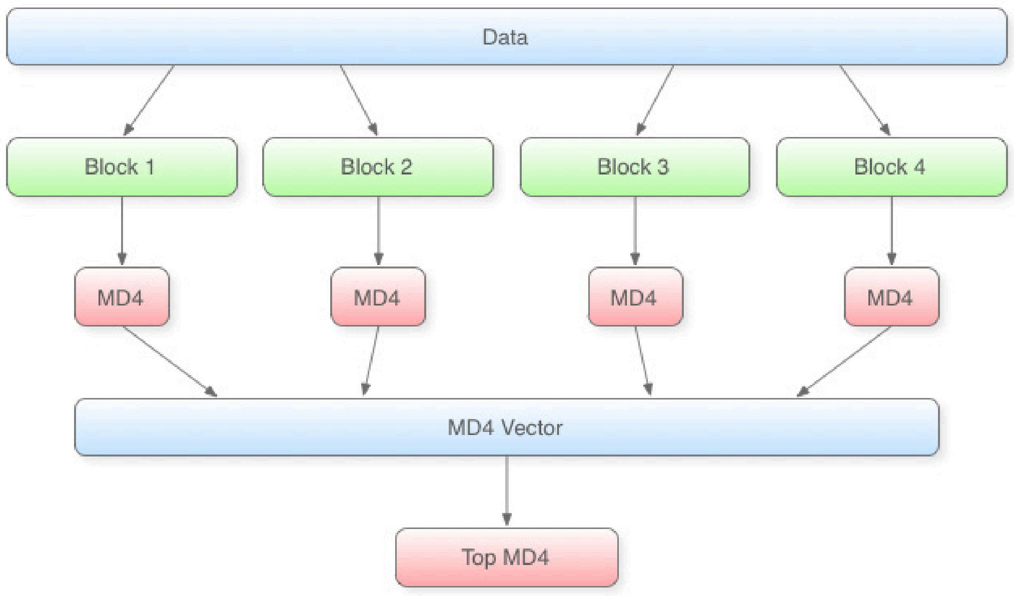


Figure 3.1: Illustration of the hashing flow in MAPAP module.

As discussed in section 3.2.1, a hash list is used. Figure 3.1 illustrates the hash flow in the MAPAP module, where the file is cut into blocks which in order are calculated into separate hashes. Finally, a combined top-hash is calculated. This will be described in detail in chapter 4.

3.2.3 Storage

OCFA has a variety of databases in its standard installation. In section 2.1, PostgreSQL and Berkeley DB is presented and both are part of the OCFA installation and they both where considered as each have their different advantages and disadvantages.

The chosen database management system is PostgreSQL. The main reason for this is the possibility to easily create a distributed module that has a central database over hashes which can be accessed by other parts of the OCFA framework. As the modules functions as separate programs, they can be run on different machines but still can access the database (see Figure 3.2).

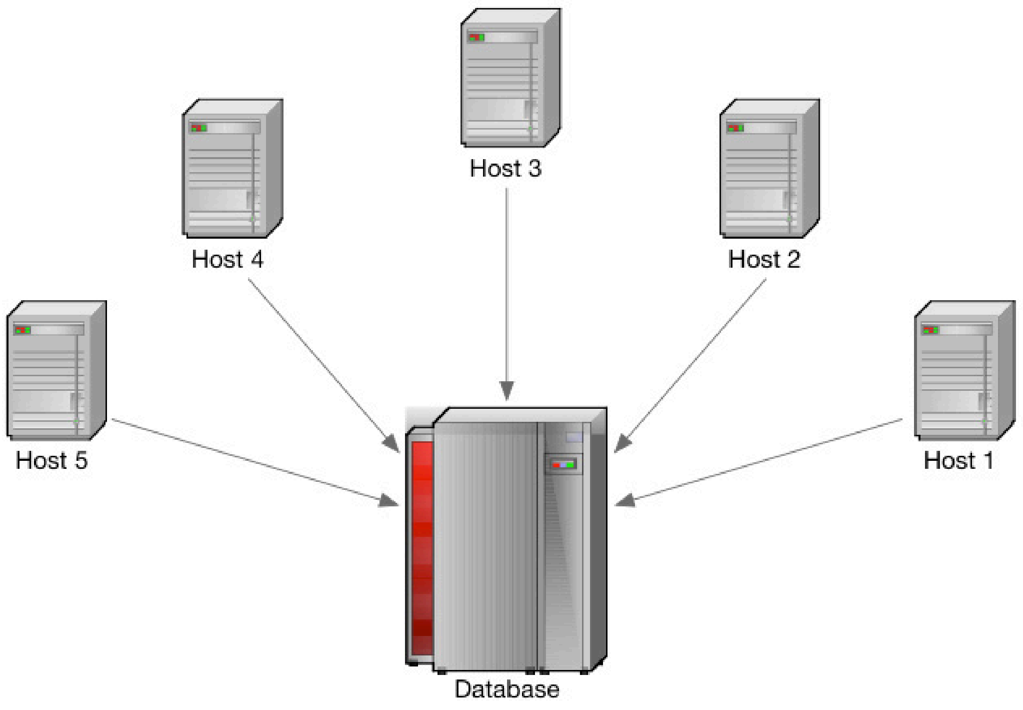


Figure 3.2: A network of five hosts connected to the same database.

Another important advantage of using PostgreSQL as the underlying database management system is the possibility to combine many queries into one. This can be used as a method of reducing load on the database. At last, the project's participants have better familiarity with database management systems with support for SQL, which reduces the workload to implement and use it.

3.2.4 MAPAP module overview

The sole objective of this project was to develop a new module for the OCFA framework. The owner of the project is FIVES which was presented i section 2.3. The aim was to implement a module with functionality to compare hashes of OCFA evidence image files with hashes provided by the MAPAP project (see section 2.4) to extend OCFA with similar functionality.

The MAPAP module is defined in OCFA as an extractor module to get access to the appropriate libraries and thus the correct successor classes. To be able to receive evidence data that the module can process, the modules main class must inherit from an object named EvidenceFileAccessor which is provided by the OCFA facade library.

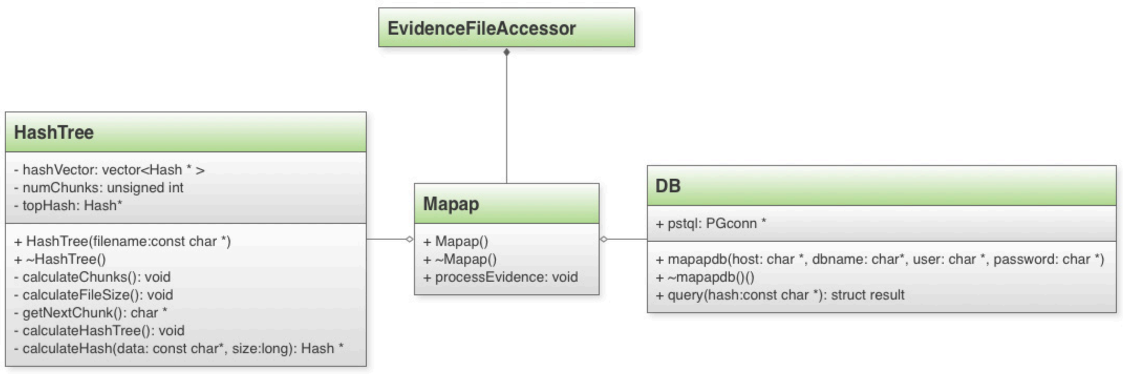

EvidenceFileAccessor
HashTree
- hashVector: vector<Hash * >
- numChunks: unsigned int
- topHash: Hash*
+ HashTree(filename:const char *)
+ ~HashTree()
- calculateChunks(): void
- calculateFileSize(): void
- getNextChunk(): char *
- calculateHashTree(): void
- calculateHash(data: const char*, size:long): Hash *
Mapap
+ Mapap()
+ ~Mapap()
+ processEvidence: void
DB
+ pstql: PGconn *
+ mapapdb(host: char *, dbname: char*, user: char *, password: char *)
+ ~mapapdb()()
+ query(hash:const char *): struct result

Figure 3.3: An overview of the MAPAP module design

As illustrated in figure 3.3, the module architecture consists of three main classes where Mapap is the junction point. It binds the classes within the module together and is also the class that connects the MAPAP module with OCFA. Mapap is the coordinator and initiates the process. It fetches the evidence through EvidenceFileAccessor and routes it to HashTree and DB to process it to get a result. When one evidence file is finished, Mapap stores the result in the evidence metadata.

The MAPAP module methods and functionality will be discussed in detail in chapter 4.

3.3 Summary

This chapter has discussed an overview of the different aspects of the MAPAP module project.

- The technological aspects as language and environment has been presented

- Choice of algorithms and storage has been discussed
- An overview of the MAPAP module project and design has been given.

In the next chapter, a more technically detailed description of the implementation will be given.

Chapter 4

Implementation

In this chapter different aspects of the implementation of the MAPAP module is covered and we go into depth of the parts that make up MAPAP. The development environment is described and discussed as well. The different interfaces exposed by the objects that are a part of MAPAP is presented and explained in detail. By explaining the internal flow of the different objects of which MAPAP is composed, the reader is given insight into the inner workings of the different objects.

There is a subsection in this chapter that covers how the sequences of calls needed for initialization and evidence processing. Further, how evidence is tagged with metadata when it is either fake, known or suspected child pornography is also explained.

In the last section, a brief description of the scripts used to populate the database with the initial data on known, suspected and fake child pornography will be given.

4.1 Interfaces between modules

This section will introduce the components of the MAPAP module in a more technical manner. An overview of the components is described in section 3.2.4 and the relation between them is illustrated in Figure 3.3.

4.1.1 HashTree

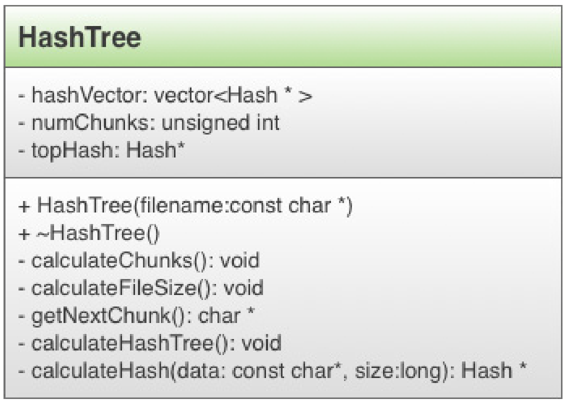

HashTree
- hashVector: vector<Hash * >
- numChunks: unsigned int
- topHash: Hash*
+ HashTree(filename:const char *)
+ ~HashTree()
- calculateChunks(): void
- calculateFileSize(): void
- getNextChunk(): char *
- calculateHashTree(): void
- calculateHash(data: const char*, size:long): Hash *

Figure 4.1: The HashTree class.

The hash tree class exposes an interface which contains two functions; one constructor and one destructor. The constructor takes a path to a file as the only argument. The destructor frees up internal memory usage and destroys the object.

4.1.2 DB

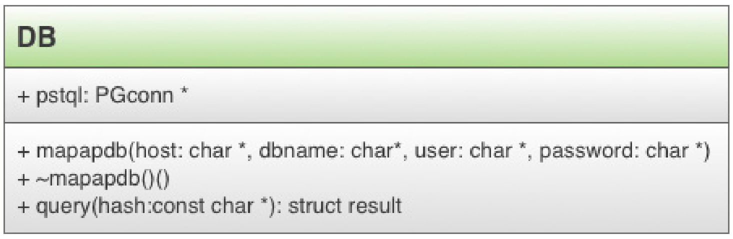

DB
+ pstql: PGconn *
+ mapapdb(host: char *, dbname: char*, user: char *, password: char *)
+ ~mapapdb()()
+ query(hash:const char *): struct result

Figure 4.2: The DB class.

The database object has an interface which contains three public functions; constructor, destructor and a query function. The constructor takes four arguments. The arguments specify the necessary parameters to successfully establish a connection to a PostgreSQL database, although the intention of the DB interface is to be independent from the specific database used in the implementation. To establish a database connection, a host name, database name, user name and password has to be provided to the constructor.

The query function takes a reference to a hash value and returns a structure which contains the result of the database query. The structure has four different values of which zero, two or four have been set to relevant information depending on which type of result the database query yielded. If the hash value cannot be found in the database all values are set to -1 in the structure to indicate there was no hit.

4.1.3 MAPAP

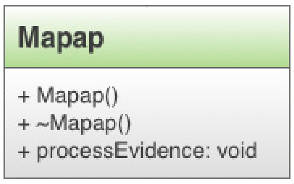


Figure 4.3: The Mapap class.

This class represents the implementation of MAPAP. It contains a constructor and a destructor which both take no arguments. The process evidence function is the main function of MAPAP. This function is called upon to process all evidence routed to MAPAP. It does not take any arguments and does not return anything.

4.2 Implementation details for hash tree

The hash tree class is the central part of MAPAP. For each evidence object that is routed to MAPAP, a hash tree has to be generated in order to calculate the top hash. The top hash is important because it is the value that is used to uniquely identify each file on the eDonkey2000 network, upon which MAPAP is used to identify harmful files.

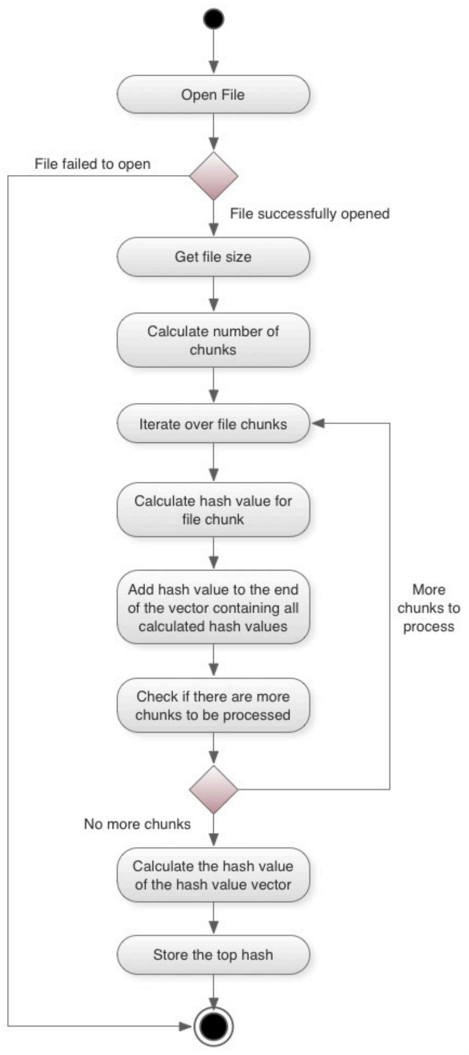

Open File
File failed to open
File successfully opened
Get file size
Calculate number of chunks
Iterate over file chunks
Calculate hash value for file chunk
Add hash value to the end of the vector containing all calculated hash values
More chunks to process
Check if there are more chunks to be processed
No more chunks
Calculate the hash value of the hash value vector
Store the top hash

Figure 4.4: Flow diagram of the hash tree.

In order to calculate the top hash value, several steps have to be taken. As illustrated in Figure 4.4, at first the evidence file must be opened, otherwise no data can be retrieved to create the vector of hash values that is needed to calculate the actual top hash.

The evidence is processed in several parts, called chunks or data blocks. The number of chunks is calculated by dividing the file size measured in kB retrieved in the previous step by 9500kB and rounding upwards.

$$NumberOfChunks = \frac{FileSize}{9500kB}$$

This means that the last chunk may only be partially filled if the file size is not an exact multiple of 9500kB. If this happens to be the case, the last chunk is padded with zeroes to fill the entire chunk. The HashTree class uses OpenSSL and its' crypto library to calculate the MD4 hash value for each chunk.

When the MD4 hash has been calculated for every available file chunk, the vector containing all the values is then converted to an array of bytes. This array is in turn passed on to the MD4 hash function in the OpenSSL crypto library. The value that is returned by the hash function is known as the hash tree's top hash. The top hash is stored for later usage in the database query function.

4.2.1 Implementation of DB front-end

The database front-end is designed to as simplistic as possible in order to efficiently create an abstract layer in between the user of the database and the actual database. The intention of this decision is that the database used shall be possible to replace if needed. This possibility is desirable if for example the underlying database is not efficient enough in terms of resource usage or if the performance is too low. The database is also an appropriate component to implement database query caches in if it also deemed necessary. Connecting to the database is one simple step.

All parameters that are required to successfully establish a connection to the database is provided to the database front-end class by passing them as arguments to the class constructor. The required parameters that must be passed to the constructor are user name, password, database name and host name.

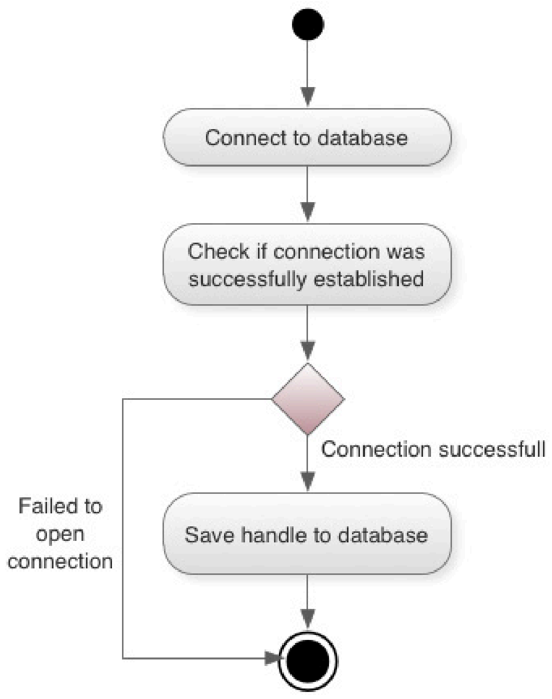

Connect to database
Check if connection was successfully established
Connection successfull
Failed to open connection
Save handle to database

Figure 4.5: Flow diagram of the DB connection.

If the connection is successfully established, the handle to the database is stored in the database front-end object for later use in every query towards the database. As shown in Figure 4.5, in the event that an error occurs during the connection establishment an exception is raised and thrown. It is up to the caller of the constructor to handle this exception and decide what mitigating measures to take.

4.3 Database queries

In order to keep the interface as simple as possible to provide a fast and simple switch between databases, only one type of query is supported. The query function takes one argument, which is a top hash. This top hash is then searched for in three different tables which contain the three different types of possible matches.

The type of the return value from the query function is a structure which has four fields; one field for each possible attribute stored in the database. The structure is filled with invalid values if no match was found in the database, or two or four values, depending on which type of match occurred (see Figure 4.6).

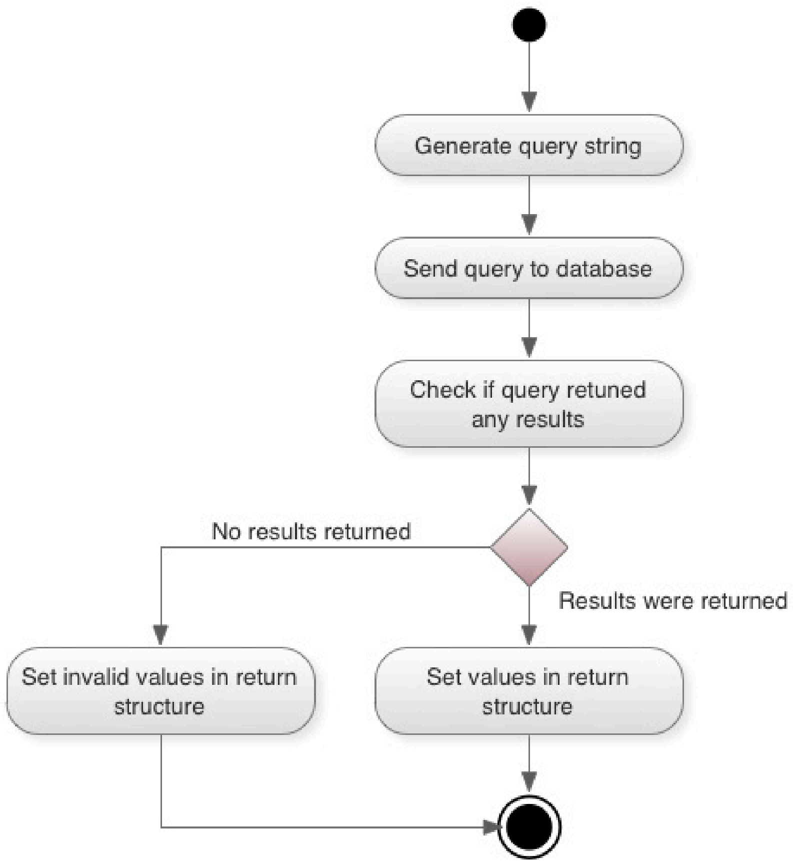

Generate query string
Send query to database
Check if query retuned any results
No results returned
Results were returned
Set invalid values in return structure
Set values in return structure

Figure 4.6: Flow diagram of DB queries.

If needed for later performance improvements this is where database query caching should be implemented.

Figure 4.7 illustrates a set of DB query sequences. This set contains the two most common sequences that will occur and they describe initialization of the MAPAP module and evidence processing, both of which are triggered by OCFA.

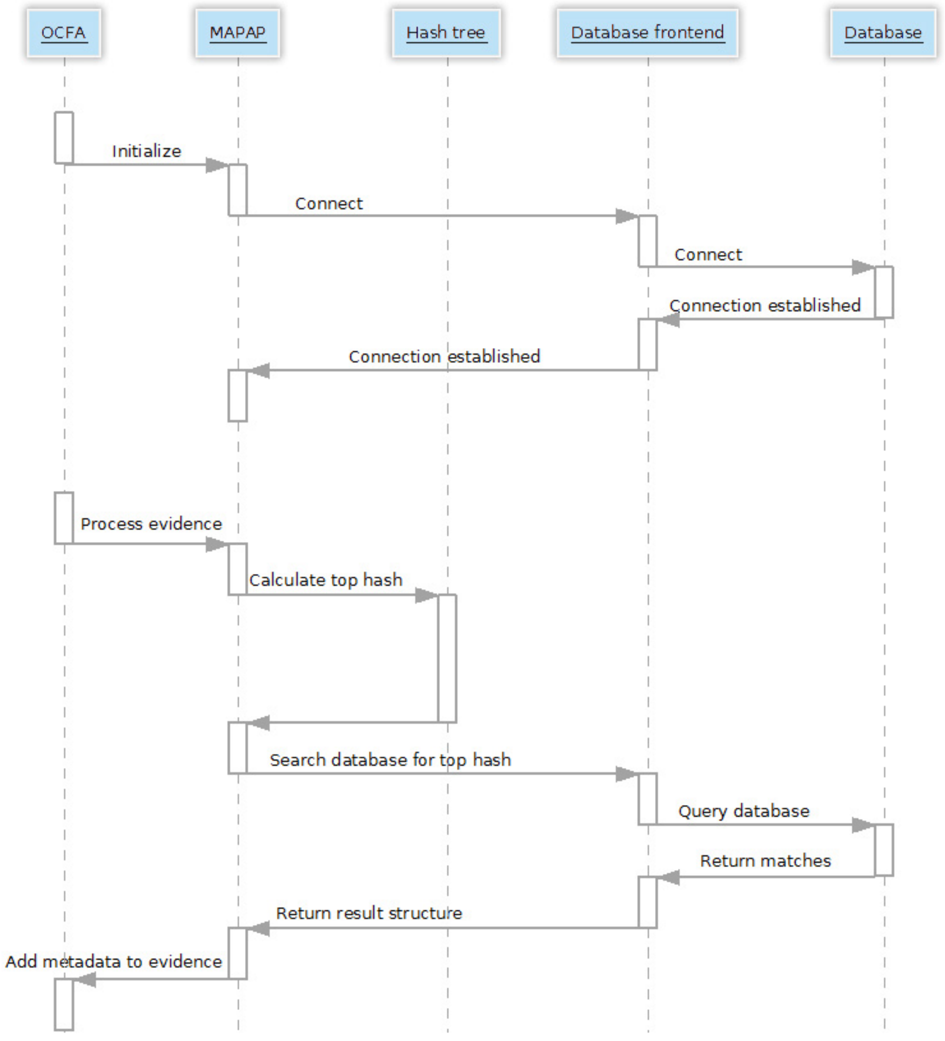


Figure 4.7: Illustration of the flow sequence in the MAPAP module.

The initialization sequence is a sequence with a small number of steps. It is, however, a very essential sequence since MAPAP will fail to start if it does not succeed. The sequence starts with OCFA requesting MAPAP to initialize itself, whereupon MAPAP tries to establish a connection towards the database. The database connection is established by the database front-end module when MAPAP creates the database front-end object. If the connection is successfully established, a handle towards the database is stored for later usage. If there is a problem in creating a connection an error is indicated by throwing an exception.

The second part of the sequence diagram depicts the sequence of messages passed between OCFA, MAPAP, the hash tree object, the database front-end and the database when OCFA places a request upon MAPAP to process a given evidence object.

By calling the MAPAP module, OCFA can start the processing of given evidence. MAPAP then passes the evidence object along to the hash tree module which calculates the top hash for the evidence. The top hash is then sent back to the MAPAP module. MAPAP relays this top hash to the database front-end. The database front-end embeds the top hash into a query which is then sent to the database. If the the top hash can be found in the database, the database front-end will fill in the corresponding values into a structure and pass this structure back to the MAPAP module. The parameter in this structure will, in the event that no matching top hash could be found in the database, be set to values indicating no match.

Depending on what is returned from the database front-end, MAPAP will set a different amount of evidence metadata. No amount of metadata is set when the structure returned from the database front-end contain nothing but invalid values. This will be the case when the evidence processed by MAPAP is not previously known child pornography. This is indicated by the absence of its' top hash in the database. If the returned structure has two valid values, the two values indicate whether the evidence is either fake or suspected child pornography and this fact will be indicated by setting two fields of metadata for the given evidence. Should the case be that four valid values are set in the returned structure, this indicates that the processed evidence is know child pornography and the four fields in the structure will be set as metadata for the processed evidence.

The sequence described in Figure 4.7 will be executed for each and every evidence object passed to MAPAP by OCFA for processing. If the performance of MAPAP is inadequate in terms of processing speed, it is possible to add a layer of buffering so that a set of top hashes can be gathered from different evidence objects and passed on to the database in one aggregated clump.

4.4 Summary

In this chapter the implementation of the MAPAP module for the OCFA framework has been presented and its' inner workings has been explained. As can be seen from the diagrams previously presented, the architecture has been kept as simple as possible to ease any future improvements, optimizations and further development of the MAPAP module if the need should ever arise.

Since short execution time is an important factor in the MAPAP module as millions of files can be processed by the OCFA framework, much effort has gone into designing and implementing the module in an execution efficient fashion. The amount of abstraction levels and inheritance hierarchies have been reduced to increase the effectiveness of the module. The cost of this comes in the form of slightly less readability of the implemented program code.

A number of places where further optimizations can be implemented has also been presented and discussed.

Chapter 5

Concluding Remarks

In this chapter, the result of the development and implementation will be explained. Further, an evaluation of the projects progress and problems will be discussed.

5.1 Result

The result of this project was a fully implemented module, the MAPAP module, in the OCFA framework. It has been tested without any indication of errors or weaknesses.

To ensure that the MAPAP module functions as intended a number of scenarios has been identified and tested against. One scenario covers the case when the top hash is to be calculated on an empty file. This file is to be padded with zeroes such that it fills one chunk. This scenario has been tested by passing empty files to MAPAP and validating the result.

In another scenario files with sizes that are not an exact multiple of the chunk size are tested with MAPAP. This requires MAPAP to correctly pad the files with zeroes before calculating the top hashes. These values have been verified.

The case where the files passed along to MAPAP have sizes of even multiples of the chunk size has also been covered. This case requires no zero padding and is correctly handled by MAPAP.

To test if MAPAP correctly allocates and frees memory it has been let run for longer periods of time and then analysed for any inconsistencies. No inconsistencies has been found in the memory management of the MAPAP module.

5.1.1 Future work

Some things have been taken into consideration as the OCFA framework is open source and other developers may come across the MAPAP module. The module has been implemented in simplest manner possible, but is still functional, so it will be easily extended.

FIVES is still an ongoing project and some modifications may be required. The module is implemented and fully functional, though some optimization of the code can always be made. As calculating the hash is the part that takes the longest time, this is something that can be evaluated and developed further.

5.2 Evaluation

The module was partially developed at the Carl Lab at Karlstad University, though much work was done remotely via a VPN-tunnel. The distribution of time was relatively evenly disposed, where the design and development of the module occupied 60% and the writing of the paper 40%.

The project can be divided into the following sections:

- Setting up and getting to know the environment
- The development of the module
- Implementation and testing
- Writing the documentation

As for getting used to the OCFA framework, the learning curve was steep, but as soon as the environment felt familiar it was friendly and easy to work with.

Writing this thesis felt dense in the beginning but in time it became a very important learning experience. At first, Open Office was used for this purpose but after some consideration a switch was made to LaTeX. This to save time but also to be able to structure the thesis better.

5.2.1 Problems

In the initial period of the project, too much time was spent on installing OCFA on the Carl Lab computers. The operating system installed did not provide with all the necessary packages to run OCFA and the OS had to be reinstalled.

Using Debian as development platform was initially not the first choice. The learning curve was steep as opposed to if Ubuntu 8.10 had been chosen. Although the OCFA project provides with specification and documentation for Ubuntu 8.10, the packages and libraries provided by that OS were insufficient. Any other Linux distribution would suffice as a development environment but small differences such as dependencies upon libraries and tools could have caused small, but not insurmountable problems. However, to save time, Debian was the choice.

Far into the project, a new OCFA was released and thus the documentation provided was removed and not immediately replaced with a new one. Once the documentation was updated, it was no longer suitable. To solve this issue, a website called Web Archive was used [3]. On this site, most of old Internet content is archived and available for browsing.

5.3 Summary

Although, the workload to pull this through was underestimated, the overall result of the project is satisfactory.

The experience and knowledge was unevenly spread between the project members and due to lack of time, the work had to be performed by the most suitable member. What we have learned from this is that more time should have been spent to planning the process before starting to develop.

One upside is to have gotten to know how a larger framework may function and how to adapt a program into that framework. It has also been very fulfilling to participate in the work of decreasing the amount of child abuse content and learn how different projects deal with this problem.

Appendix A

Acronym List

In order of appearance:

FIVES	-	Forensic Image and Video Examination Support
OCFA	-	Open Computer Forensics Architecture
MAPAP	-	Measurement and Analysis of P2P activity Against Paedophilic content
P2P	-	Peer-to-Peer
MD4	-	Message Digest 4 (Similar for MD2 and MD5)
SHA-1	-	Secure Hash Algorithm 1
SQL	-	Structured Query Language
ANSI	-	American National Standards Institute
PHP	-	Personal Hypertext Preprocessor
LISP	-	LISt Processing
DB	-	Database
BSD	-	Berkeley Software Distribution
API	-	Application Programming Interface
ACID	-	Atomicity, Consistency, Isolation, Durability
POSIX	-	Portable Operating System Interface

PoC	-	Proof of Concept
(L)GPL	-	(Lesser) General Public License
XML	-	Extensible Markup Language
VPN	-	Virtual Private Network

References

[1] Mapap information website. http://antipaedo.lip6.fr/, May 2010. Last visited in May 2010.

[2] Ofca - open computer forensics architecture library. http://web.archive.org/web/20080505054241/http://ocfa.sourceforge.net/, May 2010. Last visited in May 2010.

[3] Web Archive. Web archive. http://www.archive.org/web/web.php, May 2010. Last visited in May 2010.

[4] Freshmeat. Berkeley db. http://freshmeat.net/projects/berkeleydb/, May 2010. Last visited in May 2010.

[5] Europe's information society. Mapap information website. http://ec.europa.eu/information_society/apps/projects/factsheet/index.cfm?project_ref=SIP-2006-PP-221003, May 2010. Last visited in May 2010.

[6] RSA Laboratories. Definitions of md2,md4 and md5. http://www.rsa.com/rsalabs/node.asp?id=2253, May 2010. Last visited in May 2010.

[7] PostgreSQL. Postgresql. http://www.postgresql.org/about/, May 2010. Last visited in May 2010.

[8] The FIVES project. Forensic image and video examination support. http://fives.kau.se/, May 2010. Last visited in May 2010.

[9] FIPS PUBS. Sha-1. http://www.itl.nist.gov/fipspubs/fip180-1.htm, May 2010. Last visited in May 2010.

[10] SearchSecurity. Definition of md4. http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci527478,00.html, May 2010. Last visited in May 2010.

[11] Softpedia. Ofca - open computer forensics architecture library. http://linux.softpedia.com/get/System/Shells/

Open-Computer-Forensics-Architecture-27425.shtml, May 2010. Last visited in May 2010.

[12] Sourceforge. Ofca - open computer forensics architecture. http://sourceforge.net/apps/trac/ocfa/wiki, May 2010. Last visited in May 2010.

[13] Sourceforge. Ofca - open computer forensics architecture library. http://sourceforge.net/apps/trac/ocfa/wiki/OcfaLib, May 2010. Last visited in May 2010.

[14] Oscar Vermaas. Open computer forensic architecture, school of computer science and informatics, May 2010. University College Dublin, page 8.

[15] Forensics Wiki. Ofca - open computer forensics architecture library. http://www.forensicswiki.org/wiki/Open_Computer_Forensics_Architecture, May 2010. Last visited in May 2010.

[16] Wikipedia. Berkeley db wikipedia. http://en.wikipedia.org/wiki/Berkeley_DB, May 2010. Last visited in May 2010.

[17] Wikipedia. Definition of md4. http://en.wikipedia.org/wiki/MD4, May 2010. Last visited in May 2010.

[18] Wikipedia. edonkey 2000. http://en.wikipedia.org/wiki/EDonkey2000, May 2010. Last visited in May 2010.

[19] Wikipedia. Hash function. http://en.wikipedia.org/wiki/Hash_function, May 2010. Last visited in May 2010.

[20] Wikipedia. Hash lists. http://en.wikipedia.org/wiki/Hash_list, May 2010. Last visited in May 2010.

[21] Wikipedia. Peer-to-peer. http://en.wikipedia.org/wiki/Peer-to-peer, May 2010. Last visited in May 2010.

[22] Wikipedia. Posix - portable operating system interface. http://en.wikipedia.org/wiki/POSIX, May 2010. Last visited in May 2010.