UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Ingegneria

Corso di Laurea in Ingegneria Informatica

Forensic Analysis of Video File Containers

Candidato

Saverio Meucci

Relatori

Prof. Alessandro Piva

Prof. Fabrizio Argenti

Correlatori

Ing. Marco Fontani

Dott. Massimo Iuliani

Anno Accademico 2015/2016

To myself.

Contents

Abstract

Introduction

In an increasingly digital world, there are more and more applications where digital contents play a significant role.

Thanks to the spread of smartphones and digital cameras, the use of social networks that allow the sharing of digital images and videos is becoming more widespread. These resources, in general, contain information of personal nature; however, since these images and videos are more and more permeating our lives, their content may include information about an event that as occurred and therefore represents a significant source of evidence about a crime that can be used during an investigation.

In this context, many techniques for the analysis of multimedia content have been developed; these methods pose the goal of providing aid in making decisions about a crime so that a digital resource can be legitimately used as evidence in a courtroom.

Such is the task of the Forensic Analysis and in particular of Multimedia Forensics, i.e. to developed and apply techniques that allow, with a certain degree of accuracy, to determine whether the content of a digital resource is authentic or if it has been manipulated.

The questions that the Forensic Analyst must answer about these digital contents are:

- authenticity assessment, or whether what they represent is true and correspond to reality.

- integrity verification, or if they have been altered in such a way that does not compromise their authenticity.

- the determination of the acquisition device, a problem that in Forensic Analysis takes the name of Source Classification or Source Identification.

To give a better understanding of these problems, we provide some application scenarios.

Regarding the Source Identification, imagine a situation in which the very act of creation of an image or video constitutes a crime, due to the content of the digital resource. In this scenario, it is crucial to establish the source device that generated the offending resource, so that this resource can be used as evidence. Depending on the context, determine the origin of a digital content could mean to find the particular acquisition device or the typology of the acquisition device or establish the last processing stage, such as a compression algorithm or an editing step.

In this context, it is also important the integrity check of a digital resource. In fact, the owner of the device that created the offending digital content could change the resource in such a way that it's not possible to trace back its source.

A more particular scenario, regarding the authenticity of a multimedia resource, is when the digital content is changed so as to deceive those who are watching it. The reasons for this manipulation can be different, such as the exaggeration or limitation of the severity of an accident or disaster, as well to change the context of the situation represented.

The tools available to the Multimedia Forensics are only those that can be extracted from the digital content itself. In fact, the fundamental idea of forensic analysis of digital content is based on the observation that both the acquisition process and the post-processing step leave traces, called digital fingerprints. It is the task of the Forensic Analyst to analyze these fingerprints to determine the history and the authenticity of a digital content so that it can be used as evidence in a digital investigation.

In this regard, the analysis techniques of digital content mainly focused on two approaches. The first is the analysis of the data stream, i.e. the audio-visual signal, which is based on the research of artifacts and inconsistency in the digital content. The second is the analysis of the metadata, i.e. the determination of their compatibility, completeness, and consistency [13] with regards to the context in which it is assumed the resource has been created.

Digital Image Forensics has extensively treated the presented issues; instead, regarding the Digital Video Forensics, the research for these problems is still under study and improvement. The motivation for this discrepancy lies in the much easier way that an image can be falsified than a digital video and the several video formats (MPEG, MPEG2, H26x, VP8, etc.) in contrast to the few main formats for digital images (mostly JPEG, PNG, TIFF).

This thesis proposes a technique for forensic analysis of digital video based on *Gloe et al.* [10], whose work focuses on the study of some video formats standard and the use of the internal structure of the video file container as a tool for the forensic analysis. This choice if justified by the fact that the video file container is very fragile so that it can contain a lot of information about the history of a digital video. Both the acquisition phase and the subsequent post-processing steps or the editing of the file using various tools, modify the content and the structure of the container. This fact can be exploited both regarding Source Identification and Integrity Verification.

The thesis is divided into chapters as follows. Chapter 1 provides an introduction to Multimedia Forensics, discussing both the state of the art techniques and the standards to be followed in all aspects of a digital investigation; it also explained in greater detail the work done by *Gloe et al.* [10], giving more information on video file containers. Chapter 2 will describe in detail the structure of the video file containers for MP4-like formats. In Chapter 3, it is presented the proposed approach along with all the choices made and the details of the implemented tool. Finally, Chapter 4 will show the dataset used and explain the motivation and the organization of the experiments along with a discussion of the results obtained.

Chapter 1

State of the Art

1.1 Introduction to Multimedia Forensics

With the increasing spread of digital audio and video content, the analysis of these multimedia objects is rapidly assuming importance in the context of digital investigations, which consider both digital data and digital devices.

In digital investigations, multimedia content such as images, audio, and video are more and more being used as forensic evidence. It is, therefore, crucial to be able to extract information from such content in a reliable manner.

Multimedia Forensics has the aim to gain knowledge on a multimedia content life cycle exploiting the traces that the various processing steps leave on the data. In fact, the idea behind Multimedia Forensics is that each acquisition device and each processing operation on a digital resource leave on the media content data some traces, often called fingerprints, which characterize its history.

Many algorithms and techniques have been developed by the scientific community based on the extraction of features from the stream of audio-visual data of the multimedia content. By taking advantage of these features, such techniques try to infer information about the source/acquisition device and about which encoding and editing processes the digital resource was subject to during its life cycle. Specifically for multimedia content such as images and videos, the main approaches set themselves the goal to identify the source of digital resource and to determine if the content is authentic or has been modified from its original without any a priori information about the content under analysis. This examination is possible using just these features and tools that allow checking for the presence or absence of such features or fingerprints that are intrinsically linked to the multimedia data by the acquisition device, the encoding step, and any post-processing or editing software tools. In fact, we can distinguish three types of traces left on a multimedia content: acquisition traces, encoding traces, and editing traces.

As explained, from a scientific point of view, research has produced a large number of techniques for the analysis of multimedia content. From the point of view of the application of such methods in the courtroom, however, there is still a significant gap. This gap is probably due to a lack of communication between the legal side and the scientific side, as well as a not full maturity of the techniques that are often based on results obtained in laboratory contexts a not in real-world scenarios.

Also, there is the need for greater sharing of standard in the field of digital multimedia forensics investigations that aid Multimedia Forensics to grow and reach maturity.

Several communities and groups have worked to put together guidelines and standard on important aspects of digital investigations, such as the chain of custody, data authentication, application of the scientific method, documentation, and reporting. The ISO/IEC JTC1 Working Group 4 is one of those groups that seek to give international standards whose primary purpose is to promote the best procedures and methods for the investigation of digital evidence. It also encourages the adoption of approaches for the forensic analysis of multimedia content that are shared at an international level, in order to ease the comparison and the combination of results from different entities and organizations and also through various jurisdictions, so as to increase the reliability of such methods and the results.

Another group that aims at giving standards and guidelines for the digital investigations is the Scientific Working Group on Digital Evidence (SWGE) that deals with getting in contact different organizations that work in the field of Multimedia Forensics to promote communication and cooperation and to ensure higher quality and consistency within the forensic community.

The Scientific Working Group on Imaging Technologies (SWGIT), instead, focuses his work on image analysis technology and has the aim to facilitate the integration of such methods of analysis of images in the context of the judicial system. In fact, it provides best practices and guidelines for the acquisition, storage, processing, analysis, transmission, output image and archive of digital evidence.

Regarding the forensic analysis of images and videos, the process is defined to be composed of three main tasks: technical preparation, examination, interpretation. The technical preparation is concerned with all those operations necessary to prepare videos and images to the other tasks. The examination is the main part of the forensic analysis and deals with the application of techniques that aim to extract information from images/videos. The interpretation concerns the analysis of digital content from experts in order to provide conclusions on the features extracted from the images/videos under examination.

In this context, it becomes essential the figure of the Forensic Analyst, i.e. one who can develop and apply these methods for the analysis of digital content, interpret the results and make a summary of the results from different techniques to increase the reliability of the conclusions. It is also able to perform all of the analysis tasks following the standards shared by the forensic community. The Forensic Analyst can find the traces left on the multimedia data and acquire information on the object under examination such as who is the source/acquisition device, whether the content is authentic and if the resource is intact, and so on.

1.1.1 Applications

The major applications for forensic analysis are source identification, authentication assessment, and integrity verification of multimedia resources.

The source identification process has as objective the retrieval of information about the device of origin that generated the multimedia content under examination. It is possible to identify the source at various levels of detail. For example, sometimes it is possible to distinguish between types of sources or to make a distinction between different models of the same kind of source or between the various devices that belongs to the same type and model.

The authentication problem has to do with the task of determining whether the multimedia content is an accurate representation of an original event. The analysis process is typically base on finding inconsistency in the features extracted from the audio-visual signal.

The integrity problem concerns the task of determining whether a multimedia content has been changed or not from the moment that the acquisition device has created it. The analysis is based on the search for traces left by the editing tool or post-processing step during the life cycle that are not compatible with the source device that, for this application, is known.

1.1.2 Tools

Multimedia files can be viewed as a package composed of two main parts: the header, which contains the metadata, i.e. the information about the contents of the file; the content itself, i.e. the data stream which forms the audio-visual signal. In general, the feature extraction is based on the analysis of traces left on both the actual data and the metadata of the multimedia file.

As for the inspection of the data stream, specifically for images and videos, the examination consists of two most important aspects:

- interpretation of the content, which is the analysis of the context to understand what the data represents, who are the subjects and the objects involved, what is the environment. In general, the goal it to retrieve all the information that can be extrapolated from human observation.

- identification of sensitive details from the scene represented, such as audio-visual anomalies, the direction of the light, shadows, perspective inconsistencies, smudge marks, and so on.

Also in the context of the inspection of the audio-visual signal, a useful technique is to enhance the content, such as the improvement of the signal to detect relevant details or objects, the extraction of dimensional relationships between subjects and objects, the visual comparison between known objects and objects represented in the scene.

The other tool regards the extraction and analysis of the metadata. Metadata can be easily extracted and can contain a lot of information regarding the data stream such as source device, color space, resolution, compression parameters, date, GPS coordinates, frame rate, format tags, bitrate, sample rate, the number of channels. Obviously, the type and the number of metadata depends on the type of the file under examination and which processes have undergone during its life cycle. Once the metadata is obtained, it is the Forensic Analyst work to verify the compatibility, the completeness, and the consistency [13] of this information with regards to the context in which it is known or assumed the resource come from. In fact, metadata might be different if a digital resource originates from a social network rather than directly from the acquisition device.

1.2 Forensic Analysis of Video File Container

1.2.1 Video File Containers

The forensic analysis has focused mainly its attention on the analysis of images developing techniques that used either the data stream, the metadata, or both.

Regarding the latter, JPEG [3] and EXIF [8] metadata have become very used. Since each acquisition device and processing software use their customized quantization tables, given an image, it is possible to exploit the difference to limit the search of the origin [9]. Also, by considering the number of EXIF entries and the compression parameters, *Kee et al.* [11] associates images whose origin is not known to a certain class of source device. The forensic analysis of digital videos is currently more and more relevant. Early work has followed procedures similar to the ones used for the images, exploiting artifact and inconsistency in the data stream and examining the information extrapolated from the metadata. However, regarding the use of metadata especially for the identification of digital sources, there are concerns over falsifiability. In fact, it is just a matter of finding the right software tools, often publicly available, to easily edit high-level information such as Exif metadata. However, the known processing software and metadata editors, both for images and videos, do not have a functionality to modify low-level information, such as the internal order of the core file structures. These characteristics are thus extremely valuable and offer a higher reliability than standard metadata information.

The work of *Gloe et al.* [10] expands this idea to the video forensics by exploiting this low-level characteristic using metadata and file format information such as video file container. By identifying specific manufacturer and model characteristics, as well as traces left by processing or editing software, it is possible to assess the authenticity or the integrity of a video and to identify its origin, in terms of a particular device or a type of devices.

Gloe et al. [10] noticed that the video format standards for the data container formats prescribe only a limited number of features, thus leaving a lot of freedom to the manufacturer. The Forensic Analyst can exploit this fact as a resource, given a video, to identify the source and to assess its authenticity or its integrity.

As well as for the JPEG and EXIF low-level characteristic, the known tools do not allow you to change the video file containers since they are a core file structure. For this reason, these low-level features reduce the concerns about the falsifiability of such information as only for a subject in possess of advanced programming skills will be able to modify the internal structure and content of video file containers. Even in this scenario, it would still be very complex to preserve consistency amongst all of the metadata information, thus making the operation of falsifiability not trivial even for a highly technical figure.

Gloe et al. [10] explores in details both the AVI [12] and the MP4-like file formats. Digital cameras mainly use the AVI video stream. This thesis focused its work on the analysis of video generated by mobile phones; thus, we will give a brief overview of only the MP4-like file containers structure and the ways its characteristics can be exploited for digital forensics purposes. An explanation in great details about the structure and content of file containers of MP4-like formats will be given in Chapter 2.

Apple introduced the MOV [7] container format in 1991. Using this format as a basis, the MP4 [4] and 3GP formats have also been created. This group of file formats will be referred to as MP4-like file formats.

The video file containers of this formats are composed of atoms (sometimes called boxes) that are identified by a unique 4 bytes characters, preceded by the size of the atom. These atoms can have fields and can be nested, i.e. an atom may contain another atom. Thus, four types of atoms can be encounter:

- atoms without fields and that do not contain other atoms.

- atoms without fields but that contain other atoms.

- atoms with fields that do not contain other atoms.

- atoms with fields that also contain other atoms.

The most typical structure of the MP4-like containers is as follows:

- *ftyp* atom: it is semi-mandatory, i.e. the latest ISO standards expect it to be present and to be explicit as soon as possible in the file container.

It refers to the specific file types with which the video file, to which the container belongs, is compatible.

- *mdat* atom: it contains the data stream and specifies its size.

- *moov* atom: it is the atom with the most complex structure of the container. It is a nested atom which contains many other atoms. In this atom, and in its sub-structure, is included the metadata needed for the decoding of the data stream contained in the *mdat* atom.

This structure is not always respected. The file container of the MP4-like formats are very complex and have many elements and differences that depend on the source and the manufacturing company; thus it can constitute a valuable tool for the Forensic Analysis. The differences of containers across multiple types of video files can be found in many ways, as explained as follows:

- the relative position of the atoms: although the standard gives specifications about the location of specifics atoms (especially the main ones described above), these indications are not always respected. Above all, there is a change in position for those atoms whose position is not specified but the standard. This fact is true both at the first level of the container, which contains atoms such as *ftyp*, *mdata*, *moov*, and both concerning the atoms of the sub-structure contained in the *moov* atom.

- the presence of additional non-standard atoms at all level of the container.

- the differences in the fields values of the atoms.

All these possible differences and types of differences give rise to a large set of combinations that the Forensic Analyst can use to analyze a digital video resource.

1.2.2 Applications for Video File Container

The problem of the authentication is based on the analysis of the audio-visual signal to determine if the data is an actual represents of an original event. Thus, it is not in our domain because video file containers are treated as metadata. With metadata, you can not assess anything about the events represented in a video.

The integrity problem, instead, can be dealt with. As described above, the container of a video file is the container created by the last tool during the file lifecycle. If a video is also slightly modified by software, it will have its own container. This container may contain additional atoms or change the positions and the fields values of the other atoms compared to the file container of the video before it was processed. Therefore, if the file container content and structure of an acquisition device is known, it is possible to compare the container of the query video file with the container of a reference video file (i.e. a video generated by the known or assumed acquisition device). This way it is possible to verify the integrity of the query video file, i.e. whether the file, during the time between its generation from the source device and the present, has been modified or not.

The problem of the source classification makes sense because, since the standards define only a few mandatory features for the file container, the manufacturing companies are left with a lot of freedom. This space for interpretation means that every class of device will have its own different container structure and values. This fact can be used to construct a set of features to distinguish between devices. For examples, in the case of video created by smartphones, it can be used to determine the belonging of a video to a brand, or to a specific brand and model, or to a specific brand, model, and operating system. It becomes necessary to find a way to represent the various classes in a training set properly. Besides, a compatibility measure must be defined to compute the likelihood of a query container to belong to a particular class of devices.

The proposed approach in this thesis will do just that, giving all the theory and the necessary information of how to implement a pipeline to solve the integrity and the source identification problem of a video file using its file container.

Chapter 2

Video File Containers

Chapter 3

Proposed Approach

3.1 Overview

Our method specifically focused on Source Identification and Integrity Verification. In the following, we will explain the theory behind our proposed approach, that is the main idea of why and how to use video file containers for Multimedia Forensics.

As mentioned in the previous chapters, video file containers contains structured information about the content such as content-related metadata (acquisition time, modification time, place, etc.), number of tracks and signals (audio and video), and codec data (quantization tables, etc.) that is necessary to decode and present the signal. Since the file format standards leave room for freedom to the manufacturers about how the file container is composed, we can harness both its content and, mostly, its structure. The idea behind the use of file containers is that both the source device or the platform from which a video originates leave traces on its containers, so that we can determine its history.

For Source Identification, we are in this scenario: given a query video we want to assess if it belong to a class C based on its file container, where for a class we mean a source of origin. Specifically, during this thesis, we have focused our study on video file generated from smartphones. In this case, we can imagine the possible classes as having a hierarchic structure; so a class could be seen a specific brand, a specific model of a certain brand, or a specific operating system on a certain brand of some brand of smartphones.

Basically, given a query video and a class C, we pose a binary question: does this query video belongs to the class C? In order to answer that question, we split the ground truth set of videos in two classes X_C and $X_{\overline{C}}$, i.e. of videos belonging to class C and not respectively.

To determine which of the two classes the query video belongs, we need a compatibility score. We create Ω that is the set of all the attributes ω of the atoms contained in each file containers of the ground truth media. We determine the discrimination power of each attributes ω for the class C,

$$W_C(\omega) = \frac{\sum_{i=1}^{N_C} \mid X_i \cap \omega \mid}{N_C}$$

with $X_i \in X_C$ and N_C the number of videos of the ground truth that belongs to the class C.

Similarly, we compute the discrimination power of each attributes ω for the classe $\overline{C}$,

$$W_{\overline{C}}(\omega) = \frac{\sum_{i=1}^{N_{\overline{C}}} \mid X_i \cap \omega \mid}{N_{\overline{C}}}$$

with $X_i \in X_{\overline{C}}$ and $N_{\overline{C}}$ the number of videos of the ground truth that belongs to the class $\overline{C}$.

With the discrimination power, we can determine how significant an attribute is for a certain class.

The procedure described above is what we refer to as the training phase. For the test phase, given a query video $X = \{\omega_1, ..., \omega_n\{$, we solve the two hypothesis test problem:

$$H_0 : X \in \overline{C}$$

$$H_1 : X \in C$$

The we determine the likelihood of observing $\omega_j \in X, j = 1, ..., t$ for each of the two classes.

$$P(\omega_j | H_0) = \Omega_{\overline{C}}(\omega_j)$$

$$P(\omega_j | H_1) = \Omega_C(\omega_j)$$

Supposing, that ω_j are independently distributed we can compute a likelihood

$$L(X) = \frac{\prod_{\omega_j} \Omega_C(\omega_j}{\Omega_{\overline{C}}(\omega_j}$$

to determine whether a query video X to belong to the class C.

For Integrity Verification, the approach is simpler because we don't need to deal with classes. In this scenario, we have a query video X that supposedly come from a certain device and we want to assess if this supposition is true or if the video has undergone some other processing step during its lifecycle. In order to do that, we create a reference video Y with the device that supposedly is the source of the video X; to assess the integrity, we need to determine the compatibility of the file containers of the query video X with the file container of the reference video Y.

For each attributes of Y, we check their presence in the file containers of X; then we do the same operation in reverse, that is checking the presence of the attributes of X in the file containers of Y. The results will be a percentage representing a measure of compatibility between query and reference, i.e. by how much the two file containers under examination differs. It will be up to the Forensic Analysis, by also checking for which attributes the two file containers differ, to determine if this compatibility score is enough to assess the integrity of the query video or not.

In order to put into place the described approaches, we developed several applications that can be used both as a Java library and a command-line tool. Finally, we implemented a web application so that a user can utilize these methods more easily.

3.2 Video Format Tool

3.2.1 Features

The Video Format Tool is a software application that we have implemented mainly to extract the file container from an input video. However, this program also presents various other features which serve to manipulate file containers; these features are both implemented as low-level functions, so that they can be used as functions from a Java library, and as high-level functions, that are used as interfaces for the command-line tool.

The implemented features are:

- Parse: given input MP4-like format video file (MP4 and MOV), it extracts the file container using the *MP4Parser* library [5] and it saves it in an XML file, using the *JDOM* library [6].

- Batch parse: given as input a folder containing video files, it parses all the videos in the folder and sub-folder, saving them into XML files by recreating the same folder structure.

- Draw: it is used to draw in a window a tree, given an XML file as input that represents a video file container.

- Merge: given two XML input files, it combines them into a single XML file. By taking one as the base for the merge, it adds to it the atoms that only the other XML file has; also, for atoms that are in common, it considers the attributes and, by looking at their values, it adds them to the base XML files, so that for each attributes we will have a vector of values.

- Update: it is an advanced method to use the merge. Instead of giving two XML files as input, it takes a folder that contains XML files and merges them into a single XML as explained above. It also considers sub-folders.
- Compare: given two XML files, it compares them and it returns a measure of how much they differ.

3.2.2 Implementation

During the development of the application, we have made several implementation choices. In the following, it will be described the reason behind them, in addition to further details. The features that will be explained are only the main ones: parse, merge and appear:

3.2.2.1 Parse

The *parse* feature make use of the *Mp4Parser* library, a Java API to read, write and create MP4-like files.

To explain how the *parse* feature is implemented, we will follow the process of extraction of a file container from the input of a video to the creation of the corresponding XML file.

First, the function *parse()* is called; this function servers as an interface for the command-line application to extract a file container from a MP4-like video and parse it into an XML file. It takes two String as parameters: the first representing the path of the file video that we want to parse, and the second representing the path of the directory where to save the resulting XML files. After validating the input file path, using the *MP4Parser* library, it extracts the ISO file from the input video. Then it called the *parser()* function, passing the extracted ISO file.

The *parser()* function is used to parse an ISO file into a *JDOM Element*. Firstly, it creates a root *JDOM Element* and then it calls the constructor for the *BoxParser* class, passing the ISO file. Finally, it calls the *getBoxes()* method of the *BoxParser* object.

The *getBoxes()* method is a recursive function that takes as parameters: an *AbstractContainerBox* object, from the *MP4Parser* library, that represents an abstract base class that is suitable for boxes (i.e. atoms) of the file container that act as container for other boxes; the root *JDOM Element* created by the *parser()* function. The first time that this method is called, the first parameter will be a *Null* object. It will extract a list of children boxes from the ISO file. The boxes extracted from the ISO file will be the boxes at the first level of nesting of the file containers, generally ftyp, mdat and moov boxes. For each child box, it creates a new *JDOM Element* using as a name the identifier of the box, i.e. the 4-byte code. Then, it extracts and parse the attributes of the box and will set them as fields of the *JDOM Element*. At this point, there is the recursive step: if the box under examination is also a container of other boxes, the *getBoxes()* method will be called again this time passing the box as an *AbstractContainerBox* instead of null, and the newly created *JDOM Element*, that will act as a root for the lower levels of the file containers; if the box does not contain other boxes then the newly created *JDOM Element* is added to the root and a new child box will be parsed. At the end of the recursion, the first *JDOM Element* that we passed from the *parser()* function, will contain all the information about the file container, preserving its tree-like structures and its attributes values.

Finally, the flow of execution will return to the *parse()* function which will ensure that the *JDOM Element* will be saved in an XML file in the passed output folder.

During the development of this application, several implementations choices were made that we will explain in the following.

The parsing of the attributes for each box takes place using some wrapper functions. In fact, the *MP4Parser* library implements a *toString()* method for each specific *Box* class; however this method is not always consistent and it will not be even present for some box types. To overcome this issue, we realize a *Wrapper* interface that require the implementation of said *toString()* method. For each of the Box types that we encounter, we created the corresponding wrapper classes that implement the Wrapper interface, as well as a default *Wrapper* class that deals with unrecognized boxes. This way, we have full control over the *toString()* method and how the attributes of a box are retrieve. One way that we customized the extraction of the attributes, it is the addition of an attributes to each box; we add an attribute called count in order to determine the presence or the absence of the boxes, that will be useful afterwards, especially for boxes that do not have attributes. Also, it extensible in the sense that, if we are in the presence of a new box that we previously never encounter, we can easily implement its wrapper class and its *toString()* method.

One of the aspects that make file containers a valuable resource for the forensic analysis is the presence of low-level characteristics such as its structure and the locations of each atom in the containers. To take advantage of this feature, the name of each *JDOM Element* object will be formed by the 4-byte code of the corresponding atom along with a index number that represents the relative position with respect to the other children. This is especially useful for exploit the differences in the container structure: if two file container have a different order for, as an example, the *ftyp*, *mdat* and *moov* atoms, then, by using this indexing, we will be able to notice and harness the variation in the container structure.

Instead, the position of the attributes of each atom is not relevant. Whether it is used the toString() functions of the Mp4Parser library or the one that we implemented as classes that extend a Wrapper interface, the order of the attributes is still arbitrarily decided by either one of the two proxies. This fact means that the order has nothing to do with choices that the manufacturer made and it cannot help to determine the source device or platform of a video file.

3.2.2.2 Tree interface

In order to represent the XML files as objects, from each *JDOM Element* can be built a custom *Tree* object, so that they can be manipulated more easily accordingly to our needs.

The *Tree* interface is implemented in order to maintain the same nested structure of a XML file that represents a file containers. It is used as the *Component* class in the *Composite* pattern to implements trees. Such as a file container or a XML file, a tree can have children, which are other sub-tree, and fields, which are couples of name and value.

Each tag of an XML file can be a *Node* or a *Leaf*, both classes that implements the *Tree* interface. Each *Tree* object has a name for identification, a String variable whose value is taken from the name of the tag, a father, that is another Tree object or a *Null* object for the root Tree object, a list of children Tree objects and a list of Field objects, that correspond to the attributes of XML tag.

The *Tree* interface also requires many feature to add, remove, retrieve and modify each *Tree* object along with its children and its list of *Field* objects.

3.2.2.3 Merge

The *merge* feature combines two XML files, representing two file containers, and merges them into a single XML file. It is the features that will be used, during the training phase, to create Ω, the set of all the attributes ω of the atoms contained in each file container of the ground truth videos.

The *merge()* function serves as an interface for the command-line program to merge two XML files into a single one. It takes as parameters two *String* representing the path of the first XML file and the path of the second XML files, respectively.

Firstly, it creates two Tree objects from the input XML files; then it proceeds to call the *mergeTree()* function, by passing to it the two *Tree* objects. The first *Tree* object is taken as the base to which the elements of the second *Tree* object will be added. The function begins by extracting the children of the second *Tree* object; for each child, it checks its presence in the first *Tree* object. If it is not present it will add that child *Tree* object to the first *Tree* object at the corresponding level and by setting the right father. If it is present, it will check the fields of the corresponding child in the first and second *Tree* object. For each field couple, it will compare their values and, if they differ, it will add it to the values of that particular field for the first Tree object. This way the first Tree object will have fields values that represent vector of values.

The function now proceeds recursively; the *mergeTree()* function will be called again by passing the corresponding child in each of the *Tree* objects. The recursion will stop once it will have reach the *Leaf* object for each of the tree branches.

The final results will be contained in the first *Tree* object passed, that now represents the union of the two XML files, both in terms of atoms and of attributes values.

Then, the merge() function will save the resulting merged *Tree* object into an XML file.

Since the result is of *mergeTree()* in another *Tree* object, this features can be used repeatedly to merge many XML files by adding to the same merged one. This merged XML files represents the Ω set.

3.2.2.4 Compare

The *compare* feature serves to compare two XML files and give a measure of how much the file containers they represent differ. It is the feature used to verify the integrity of query video by confronting it to a reference video for which the source device is known.

Similar as for the other features, the *compare()* function serves as an interface for the command-line program. It takes as parameters two *String* representing the path of the first XML file and the path of the second XML files, respectively.

The first input XML file represents the video that will be taken as a reference and the second input will represent the query video. Both XML files are converted into two Tree objects and are passed to the *compareTree()* function. The *compareTree()* is a recursive function that behaves similarly to the previously mentioned recursive functions: it iterates through the reference Tree object children and checks for difference for each of the corresponding child in the query video. If there is not a corresponding child, then all the attributes of the child of the reference Tree object will be counted as difference. If there is a corresponding child, it will be counted a difference every time that a pair of attributes from the reference Tree object and the query Tree object differs in their values. Also, attributes for which a difference is found are saved in a list, along with the corresponding values from both of the Tree object.

Besides the number of differences, it is also counted the total number of attributes of the reference Tree object. The final results will be a percentage, computed dividing the number of differences found with the total number of attributes, that represents how much the two file containers differ.

The results are then returned as JSON formatted output.

3.2.3 Command Line Tool

In the following section, we will explain how to use the command-line interface for the Video Format Tool by giving some examples of usage.

- Extract a file container from a video and save it into an XML file.

```
vft --parse -i input.mp4 -o /output_folder
```

- Batch parse a directory of videos. It also recreates the same sub-folders structure.

```
vft --batch -i /input_folder -o /output_folder
```

- Draw a tree from an input XML file.

```
vft --draw -i input.xml
```

- Merge two XML files, with or without consider the attributes.

```
vft --merge -wa -i input.xml -i2 input2.xml -o /output_folder
```

- Merge all XML files in a given directory into a single XML file saved in the output folder, with or without attributes. It also considers XML files in sub-directories.

```
vft --update-config -wa -i /input_folder -o /output_folder
```

- Compare two XML files and return a measure of how much they differ.

```
vft --compare -i input.xml -i2 input2.xml
```

- Print the help message.

```
vft --help
```

3.3 File Origin Analysis Tool

3.3.1 Functional Requirements

The File Origin Analysis Tool is the software application implements the theory behind our proposed method regarding Source Identification based on video file containers and it is developed to be used as a command-line program.

The main features are:

- Training: given two sets of videos, one representing a class C and the other a class $\overline{C}$, it will creates Ω, saved as a XML files. Then, for each attribute ω it will compute the discriminant power regarding the class C and the class $\overline{C}$, respectively.

- Test: given a query video X, for each attribute $\omega_j \in X$ it will check the corresponding discriminant power estimated by the previous training phase for both the class C and the class $\overline{C}$. By combining all the discriminant powers of all the attributes ω_j, it will compute a measure, representing the likelihood that X belongs to the class C.

3.3.2 Implementation

3.3.2.1 Training

The task of the training phase it is to compute the discriminant power of each attribute $\omega \in \Omega$. The work-flow of the feature is as follow. The *Train* class is the one that deals with the training phase. Its constructor will take as arguments two *VideoClass* objects, one for the class C and the other for the class $\overline{C}$, that will contains a list of XML files for each ground truth video of the class, and a *String* for the output folder in which the XML file that represents Ω will be saved. Using the *train()* method of the *Train* class, it will first iteratively use the merge feature of the *Video Format Tool* library to compute Ω and represent it as a XML file, that will be referred to as the configuration file. The resulting XML file will contain all of the atoms found in the file container of the ground truth videos. For each attribute, as explained in the Video Format Tool section, its value will be a list of possible values found for the attribute of that particular atom in every file containers of the ground truth. Also, to each attribute will be associated two vectors of weights, the same size as the number of possible values, initialized to zero. These vectors will be used to compute the discriminant power of each attribute value regarding both class C and class $\overline{C}$.

This computation is done by the *computeWeights()* method of the *Train* class, first for the class C and then for the class $\overline{C}$. It takes two *Tree* object of the *Video Format Tool library*, one representing the configuration XML file and the other a video of class C. For each atom of the configuration *Tree* object, it will search for the corresponding atom in the class C *Tree* object. Then, for each attribute of the class C *Tree* object it will check for its value in the list of possible value for that attribute contained in the configuration *Tree* object and will increment the weight that correspond to that value. Each weight w_i is normalized by the number of videos of the class under examination, so that $w_i \in [0, 1]$.

Finally, the configuration Tree object will contain the discriminant powers of every attributes values for each atom for the ground truth with regard to the class C. The same process will be applied to the videos of the class $\overline{C}$ and then both configuration Tree object will be saved in two separate XML files, one for class C and one for class $\overline{C}$, terminating the training phase.

3.3.2.2 Test

In the test phase, given a query video X, we want to determine, based on its file container, if X belongs to the class C or not. In order to do so, we need a likelihood measure that takes into account all the discriminant powers of the attributes value of the file container of the video X with regards to class C and class $\overline{C}$.

The constructor of the class *Test* takes as arguments the path of the XML file, that represent the file container of the query video, and the paths for the configuration XML files for both class C and class $\overline{C}$. The method *test()* will build all passed XML files to *Tree* objects. Then, it will create a *StandarLikelihood* object, a class that implements the *Likelihood* interface. In this way, we can change and customized how we want the likelihood to be computed.

The implemented *StandardLikelihood* works as follow. Using the *computeLikelihood()* method, it explores all the atoms of the query Tree object. For each of these atom, it will search for the corresponding one in both configuration Tree objects. For each attribute of the atom under examination, it will compute the ratios between the weights from both classes associated with that value, with the weight for the class C as a numerator and the weight for the class $\overline{C}$. All the ratios for each attribute of the query file container will then be multiplied between them, obtaining a likelihood measure $L(X)$, which is then smoothed with the natural logarithm. The final result $l(X) = ln(L(X))$ can be used to determine whether X belongs to the class C.

3.3.2.3 Implementation Details

The list of possible values associated to an attribute of the configuration files is computed by merging the values for that certain attribute found both in the videos of the class C and the class $\overline{C}$. It is possible that a given value for an attribute is only present in the videos of a class but never in the other. This fact means that the weight, i.e. the discriminant power, associated to that attribute value will be zero for the latter. Also, the origin of the query video could be unknown, meaning that its source will differ from all the video of the ground truth. The file container of the query video could have both atoms and attributes values that are not present in the configuration files. When computing the ratio for these new attributes, it follows that they will not have a weight associated.

To deal with these issue, we have identified four different scenarios that can happen when computing the ratios between the values weights.

The default case is the easiest one and it occurs when both the weight associated to class C and the weight associated to class $\overline{C}$ are greater that zero.

When the weight associated to the attribute value for the class $\overline{C}$ equals to zero, then the ratio between the weights will have the form of $\frac{w_i}{0}$, with w_i the weight associated to the attribute value for the class C. A division by zero is not acceptable. To overcome this issue, we modified the computation to

$$\frac{w_i}{\frac{1}{N_C + 1}}$$

, where N_C is the number of videos in the class C. In this way, the denominator will have the smallest possible value and the ratio will still pull the likelihood towards the class C.

Instead, when the weight associated to the attribute value for the class C equals to zero, the ratio will have the form of $\frac{0}{w_j}$, with w_j the weight associated to the attribute value for the class $\overline{C}$. In this case, since the numerator is zero, the ratio will equal to zero too. This is an unwanted scenario because, since all the ratios are multiplied between them, a single ratio that equals to zero is enough to make the general likelihood zero too. We decided to solve this problem by modifying the computation of the ratio for this case to

$$\frac{\frac{1}{N_{\overline{C}} + 1}}{w_j}$$

, where $N_{\overline{C}}$ is the number of videos in the class $\overline{C}$.

Finally, when an atom or an attribute value is not present in the configuration files, we have the case where both weights equal to zero. Since we want to determine if a given query video X belongs to the class C, any information on the file container that is not present in the configuration files for the class C will be treated as if it means that the video does not belong to the class C, even if the same information is also not present in the configuration file for the class $\overline{C}$.

When multiplying the ratios for a certain atom, the resulting value can be seen as a likelihood of observing that specific atom with its attributes values in either of the two classes. However, the values of the attributes for an atom are not, in general, independently distributed. This means that the values of some attribute can change in groups. If we simply multiplied all the ratios, it will be as if we are counting the same information multiple time, pushing the likelihood towards the class C or the class $\overline{C}$ improperly.

To solve the issue, we multiply the likelihood ratios of each atom by also taking into consideration the decorrelation factor. Given a vector of likelihood ratios $(x_1, \ldots, x_n)$ we compute the likelihood

$$L(\overline{x}) = \prod_{i=1}^{n} x_i^{\alpha_i}$$

with

$$\alpha_i = \frac{(n-1)\gamma_i + 1}{n}$$

$$\gamma_i = -\frac{n}{logn} P(x_i) log P(x_i)$$

and where $P(x_i)$ represent the probability of finding that value of ratio in the vector. We have three cases:

1) $P(x_i) = \frac{1}{n}$ with $x_i \neq x_j$, $\forall i \neq j$. Then we will have

$$\gamma_i = -\frac{n}{logn}\frac{1}{n}log\frac{1}{n}$$

resulting in $\alpha_i = 1$. The likelihood will be computed as

$$L(\overline{x}) = \prod_{i=1}^{n} x_i$$

2) $P(x_i) = 1$ with $x_i = x_j$, $\forall i, j$. Then we will have $\gamma_i = 0$ and $\alpha_i = \frac{1}{n}$. The likelihood will be computed as

$$L(\overline{x}) = \prod_{i=1}^{n} x_i^{\frac{1}{n}} = x_1^{\sum \frac{1}{n}} = x_1$$

3) $x_i = x_j$, $i, j = 1, \ldots, k$ with $P(x_i) = \frac{k}{n}$ for $i = 1, \ldots, k$ and $P(x_j) = \frac{1}{n}$ for $j > k$. Then we will have

$$\gamma_i = -n \frac{n}{logn} \frac{k}{n} log \frac{k}{n} = k(log \frac{k}{n}) logn$$

and

$$\alpha_i = \frac{(n-1)(klog\frac{k}{n})logn + 1}{n}$$

The likelihood will be computed as

$$L(\overline{x}) = x_i^{\sum \frac{1}{n}} = x_1^{\left[(n-1)(klog\frac{k}{n})logn+1\right]\frac{k}{n}} \cdot \prod_{j=k+1}^{n} x_j$$

3.3.3 Command Line Tool

In the following section, we will explain how to use the command-line interface for the File Origin Analysis Tool by giving some examples of usage.

- Compute the configuration files for given classes C and $\overline{C}$.

```
foa --train --listA classC.json --listB classNotC.json -o
    /output_folder
```

- Given two classes C and $\overline{C}$, test if a video X belongs to class C.

```
foa --test -i input.xml --configA configC.xml --configB
    configNotC.xml
```

- Print the help message.

```
vft --help
```

3.4 Web Application

To facilitate the application of the proposed approach for forensic analysis using the tool previously explained, we have also implemented a web application. Its task is to act as a graphical user interface and allow a user to customize the query and present the results in a more readable format. The web application was developed using the *Javascript* runtime *Node.js* [2] and with the framework *Express.js* [1] for the server's back-end.

Per utilizzare più semplicemente gli strumenti di analisi forense implementati e spiegati precedentemente, è stato deciso di implementare una web application che funga da interfaccia e permettere ad un utente di customizzare le query e ottere un output dei risultati. La web app è sviluppata come node.js app ed utilizza un server express. Segue quindi la struttura: html + css per aspetto, js client-side per interattività e ajax call, js server-side per rispondere alle chiamate ajax e eseguire le features e poi rispondere al client (node+epress).

Le features principali sono classify, per identificare la classe del dispositivo sorgente dato un video, compare, per integrity analysis, infine test per rendere più veloci i test con query.

Il dataset utilizzato è diviso in training e testing. Ciascun video possiede un file xml di informazioni utilizzato per costruire il groundtruth. Tali informazioni, insieme ai path del video, dell'info.xml e del file xml del container, sono salvati in un database. Tale database viene interrogato per ottenere i video per il training e per ottenere i video da testare.

3.4.1 Features

La scelta del campione della voce anonima, dopo averne accertato l'idoneità al confronto, influenza il risultato ma più ancora lo influenza la scelta della popolazione di riferimento. Sorge il problema, in parte analogo a quello relativo alla scelta delle voci nella composizione di un line up: bisogna decidere se la popolazione di riferimento debba essere un campione dell'intera popolazione dei parlanti oppure un campione di parlanti aventi caratteristiche simili a quelle della voce anonima: ad esempio stesso ambiente sociale, stesso sesso, stesso livello culturale, stessa parlata regionale, ecc.

É possibile seleziona la features dal nav in cima alla pagina, con in aggiunta una pagina dove spiega come usare tali features.

- Classify: questa features funziona in due modalità per il training, ovvero manuale e automatica. La modalità manuale si ha quando l'utente è a conoscenza del dispositivo sorgente del video query ma non ha il dispositivo ed intende verificare tale classe. La modalità automatica invece viene utilizzata quando l'utente è totalmente all'oscuro del dispositivo sorgente del video query. Le due modalità funzionano così:

- manual: nel box class l'utente seleziona il brand, il model, e il sistema operativo. Questi elementi sono in sequenza; l'utente può scegliere solo il brand oppure il brand e il model oppure tutti e tre. In base a tale scelta cambierà come la classe avversaria (B) verrà selezionata, ed anche quella scelta (A). Per scegliere la classe A viene fatta un query al database sulla tabella dei video di training in base alla scelta della classe. Se scelgo solo il brand prendo tutti i video di quel brand; se scelgo brand e model prendo tutti i video di quel brand e specifico modello; infine se scelgo brand model e os prendo tutti i video di quel brand e modello con quel sistema operativo specificato. La classe B viene presa sulla base della scelta per la classe A. Se è stato scelto solo il brand prendo tutti i video i cui dispositivi non appartengono a quel brand; se è stato scelto solo brand e model, prendo tutti i video provenienti dai dispositivi di quel brand ma degli altri modelli disponibili per quel brand; infine se è stato scelto brand model e os, prendo tutti i video provenienti dai dispositivi di quel brand e modell ma degli altri sistemi operativi disponibili. Nel caso di brand model e os, se non sono disponibili altri sistemi operativi mi riporto al caso 2, ovvero brand e model. Nel caso di brand model, se non sono disponibili altri modelli per quel brand, mi riporto al caso 1, ovvero brand.

- automatic: nel box class l'utente non seleziona nulla, lascia tutto any. A questo punto verranno testate tutte le classi disponibili nel database. Verrà viene seleziona ogni possibile classe A e di conseguenza ogni possibile classe B.

Nel box upload, viene seleziona il video query che di cui si vuole predire la sorgente. È possibile caricare un video (mp4 o mov) o più video (max 5?), oppure direttamente i file xml dei rispettivi container. Nel caso in cui si caricano i file video, verrà fatto il parse del video prima di procedere.

I file di training sono creati ed aggiornati periodicamente indipendentemente dalla query. Infatti tali file dipendono dai video presenti nel dataset; la scelta della classe da parte dell'utente serve solo a decide quali file di training utilizzare. Quindi è possibile ottimizzare creandoli prima e andando a selezionare quelli che mi servono di caso in caso.

Una volta che ho caricato il video, scelto la classe A e B, il server utilizza i corretti file di training. Dopo c'è la fase di testing (utilizza foa), che, dati i video caricati, calcola la likelihood rispetto alla classe selezionata. Per il caso automatic verrà calcolata la likelihood per tutte le classi; tali likelihood vengono ordinate dal migliore al peggiore.

Infine, finito tutto, nel box output viene mostrato il nome del video query e la classe testata insieme alla likelihood associata; per il caso automatic vengono mostrati, per ciascun video caricato, i migliori 5 risutalti (classe + likelihood).

- Test: serve per fare il classify più velocemente, senza dover caricare ogni volta i video. C'è solo il box class dove selezionare la classe da testare (manual o auto)e il box output. I video da testare vengono presi dal database dalla tabella dei video di testing, dove sono già parsati in file xml. Oltre all'output come prima, viene mostrata una statistica sui risultati della classificazione per tutti i video considerati (true positivi ecc, correct classification rate); infatti in questo caso è nota la classe vera dei video testati, classe (label) che viene mostrata accanto al nome del file video testato.

- Compare: il compare sfrutta la feature compare di vft. É possibile caricare due video, uno reference l'altro query dal box upload. Una volta finito, nel box output verranno mostrati i risultati, ovvero il numero totale di attributi di reference e le differenze rispetto a query; inoltre lo stesso calcolo è fatto invertendo ref e query, per dare una visione migliore delle differenze ed eventuali somiglianze/uguaglianze.

- vft-parse: se l'utente non è in possesso dei file container, può caricare un video; se però il file video è grande e non si può/vuole aspettare il caricamento sul server, dal box upload può essere scaricato un programmino jar chiamato vft-parse. Tale programma, con una semplice interfaccia in java swing, è una costola di vft completo e permetto di fare il parsing di un video selezionato o di un intera directory. In questo modo è possibile poi caricare i file xml invece dei file video, velocizzando notevolmente il processo per tutte le features.

- problema del parlante: il motivo per cui vengono scelte le classi avversarie nel modo spiegato precedentemente è che si va incontro al problema del parlante. Il problema del parlante consiste nel fatto che ad esempio nel riconoscimento del parlato, se uso un dataset molto vario in cui includo tutte le lingue e dialetti del modo per fare il training, poi non sono in grado di classificare accuratamente un parlato query; mentre se invece per fare il training seleziono dialetti simili allora sono in grado di classificare correttamente. Ciò sembra contro-intuitivo per la realtà, dove accade il contrare, ma in questi contesti usare cose simili significa dare molto più peso a piccole differenze che nel caso globale andrebbero perse. Questa è l'idea usata alla base della scelta delle classi avversarie, che cercano di essere diverse ma simili alla classe selezionata dall'utente.

Chapter 4

Experiments and results

4.1 Integrity Analysis based on File Container

4.2 Model Identification based on File Container

4.3 Application on Social Network

Conclusions

Bibliography

[1] Express.js. *http://www.http://expressjs.com.*

[2] Node.js. *https://nodejs.org.*

[3] I. 10918-1. Information technology. digital compression and coding of continuous-tone still images. *ITU-T Recommendation T.81*, 1992.

[4] I. 14496. Information technology. coding of audio-visual objects, part 14: Mp4 file format. 2003.

[5] Apache. Java mp4 parser. *http://www.github.com/sannies/mp4parser.*

[6] Apache. Jdom. *http://www.jdom.com.*

[7] I. Apple Computer. Quicktime file format. 2001.

[8] J. Electronics and I. T. I. A. J. CP-3451. Exchangeable image file format for digital still cameras: Exif version 2.2. 2002.

[9] H. Farid. Digital image ballistics from jpeg quantization: a followup study. *[Technical Report TR2008–638] Department of Computer Science, Dartmouth College, Hanover, NH, USA*, 2008.

[10] T. Gloe, A. Fischer, and M. Kirchner. Forensic analysis of video file formats. *Digital Investigation*, 11, Supplement 1:S68 – S76, 2014. Proceedings of the First Annual {DFRWS} Europe.

[11] E. Kee, M. K. Johnson, and H. Farid. Digital image authentication from jpeg headers. *IEEE Transactions on Information Forensics and Security*, 6(3):1066–1075, Sept 2011.

[12] M. D. Network. Avi riff file reference. *http://msdn.microsoft.com/en-us/library/ms779636(VS.85).aspx.*

[13] A. D. Rosa, A. Piva, M. Fontani, and M. Iuliani. Investigating multimedia contents. *2014 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, Oct 2014.

Credit