Digital Investigation 18 (2016) 1–7

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Integrity verification of the ordered data structures in manipulated video content

Jieun Song [a], Kiryong Lee [a], Wan Yeon Lee [b], Heejo Lee [a, *]

[a] Korea University, South Korea
[b] Dongduk Women's University, South Korea

ARTICLE INFO

Article history:
Received 12 November 2015
Received in revised form 31 May 2016
Accepted 6 June 2016
Available online 16 June 2016

Keywords:
Digital forensics
Data structure
Video forgery
Integrity verification

ABSTRACT

Video content stored in Video Event Data Recorders (VEDRs) are used as important evidence when certain events such as vehicle collisions occur. However, with sophisticated video editing software, assailants can easily manipulate video records to their advantage without leaving visible clues. Therefore, the integrity of video content recorded through VEDRs cannot be guaranteed, and the number of related forensic issues increases. Existing video integrity detection methods use the statistical properties of the pixels within each frame of the video. However, these methods require ample time, because they check frames individually. Moreover, the frame can easily be replaced and forged using the appropriate public software. To solve this problem, we propose an integrity checking mechanism using the structure of ordered fields in a video file, because existing video editing software does not allow users to access or modify field structures. In addition, because our proposed method involves checking the header information of video content only once, much less detection time is required compared with existing methods that examine the entire frames. We store an ordered file structure of video content as a signature in the database using a customized automated tool. The signature appears according to the video editing software. Then, the suspected video content is compared to a set of signatures. If the file structure matches with a signature, we recognize a manipulated video file by its corresponding editing software. We tested five types of video editing software that cover 99% of the video editing software market share. Furthermore, we arranged 305,981 saving options for all five video editing suites. As a result, we obtained 100% detection accuracy using stored signatures, without false positives, in a collection of 305,981 video files. The principle of this method can be applied to other video formats.

Introduction

Vehicle accidents are an unavoidable part of our daily lives. When we become involved in a vehicle accident, there is no doubt that one of the most important problems, with the exception of saving lives, is clarifying responsibility for the accident, including any legal issues. In the last decade, Video Event Data Recorders (VEDRs) have been used as effective and trustworthy witnesses to vehicle accidents and even other criminal incidents. VEDRs are also known as Car Dashboard Camcorders, Vehicle Road Dash Video Camera, Car Black Boxes, and Driving Recorders. In this paper, we refer to such devices collectively as VEDRs. The installation and use of VEDRs is increasing rapidly, and there are a number of countries that mandate the installation of such devices. However, sophisticated video editing software makes it easy to manipulate video content without leaving visible clues. Therefore, it is not possible to be entirely convinced of the integrity of video content

* Corresponding author.
E-mail address: heejo@korea.ac.kr (H. Lee).

http://dx.doi.org/10.1016/j.diin.2016.06.001
1742-2876/

recorded by VEDRs (Poisel and Tjoa, 2011; Lee et al., 2015). Some efforts have been directed at ensuring the integrity of video content in digital images and video files in advance (Hu et al., 2015). However, such technologies have a limitation wherein the video recorder system must be pre-processed when it is established, for example, by calculating hash values, embedding watermarks into the video, and so on. Therefore, digital video forensics has become a popular technique for video integrity verification without pre-registration or pre-embedded information; this process is referred to as passive-blind video forensics.

Existing methods mainly focus on digital still images. Zheng used the statistical anomalies of each pixel in an image (Zheng et al., 2015; Rad and Wong, 2015). Qu used the statistical characteristics of image compression (Qu et al., 2014; Cao et al., 2014). Geradts checked the trace caused by a camera color filter and charged-couple device (CCD) (Geradts et al., 2001). Carvalho examined the geometric relationship between an object and light (Carvalho et al., 2015). The principles behind these methods can be applied to video content. However, such evidence can easily be replaced and forged in the image source using publicly available software (Sencar and Memon, 2008). In addition, existing methods must check entire frames in order to detect video forgeries (Wang and Farid, 2007). On an average, 1 min of video content contains 1800 frames, and thus these methods must repeatedly check all 1800 frames. If the video content has low resolution, the detection rate for these methods is reduced significantly.

To address these problems, we propose a mechanism for detecting the modification of video content with video editing software. We focus on the ordered data structures in a video file for integrity verification. The internal ordered data structures are particularly valuable and contain distinct information on video editing software authentication. Existing video editing software and metadata editors do not modify the file's data structures (Gloe, 2012). Therefore, data structure information is reliable for checking video integrity. In addition, this method requires relatively little time for detection, because it simply checks the header information of video content a single time. Moreover, this method is not affected by the resolution, because it checks the data header information exclusively.

We investigate the video file structure characteristics for each type of video editing software that would leave traces from processing the video editing software. Because such traces are an inherent characteristic of each respective video editing software suite, we can detect the specific video editing software used to manipulate the video, in addition to whether the video was, indeed, manipulated. To evaluate the accuracy of this technique, we examined 296 unmodified Audio Video Interleave (AVI) video files. We performed this examination using popular versions of video editing software, namely, Adobe Premiere CS3, Adobe Premiere CS4, Adobe Premiere CS5, Adobe Premiere CS6, Adobe Premiere CC, Sony VEGAS 9, Sony VEGAS 10 Sony VEGAS 11, Sony VEGAS 12, Sony VEGAS 13, Edius 6, Edius 7, Avid Media Composer (Avid MC) 5, Avid MC 6, and Avid Studio 1. These software suites comprise 99% of the video editing software market share. (Notably, we excluded Final Cut, because it does not support the AVI format without an additional plugin.) Although the same video editing software is used in certain cases, depending on the rendering option, the files can appear to have different ordered data structures. Therefore, we arranged 305,981 saving options for all five video editing software suites.

As a result, we found that the AVI data structures in modified video files appear consistently according to each video editing software suite. Each resulting data structure is unaffected by the original video file structure. These ordered data structures are stored in a database as the signature of each type of video editing software. Moreover, given the need to include other video editing software, the database can be extended easily. We used our own customized file parser to read, extract, and detect video content. As a result, we obtained 100% detection accuracy using the stored signatures, with no false negatives (FNs) or false positives (FPs) in our experimental environments. Using the proposed method, we can check the integrity of suspected video, and we can also find the video editing software used for the manipulation.

The main contribution of this paper is the proposal of a method for detecting the integrity violator of video content. To the best of our knowledge, this work is the first attempt to generate a video editing software signature database. Our method does not require pre-processing, and its detection accuracy is not affected by the resolution.

In the following section, we explain the general AVI file format structures. Then, we introduce the practical test setup and explain the proposed signature algorithm. Based on this method, we analyze the original video contents and store video editing software specifications into a database. Using this database, we then describe our evaluation of its detection accuracy. Finally, we summarize our investigation and provide a discussion of the experimental results.

Background

The majority of the multimedia formats used in VEDRs are AVI files. The second most common format is MP4. Only a few test devices use WMA containers. Nevertheless, the proposed method can be applied for AVI, MP4 and WMA formats. Without loss of generality, we discuss the case of the AVI format in this paper. For clarification, we explain the general AVI file format structure.

General AVI file format structure

AVI is a multimedia container format introduced by Microsoft in November 1992. AVI files can contain audio and video data. A ″chunk” refers to a fragment of information that is used in many multimedia formats, such as MP3 and AVI. Each chunk is identified by a four-byte delimiter. The four-byte delimiter is called a Four-Character Code (FourCC) tag. This tag is used to identify the stream type, data chunks, index entries, and other information. An AVI file takes the form of a single chunk in an RIFF formatted file, which is then subdivided into chunks. Every chunk has the following basic structure (see Fig. 1): the ChunkID contains four ASCII characters that identify the chunk field name; the ChunkSize is the length of the data stored in the ChunkData field, excluding any padding areas;