Design Tradeoffs for Developing Fragmented Video Carving Tools

By

Eoghan Casey and Rikkert Zoun

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2014 USA

Denver, CO (Aug 3rd - 6th)

Digital Investigation 11 (2014) S30–S39

Contents lists available at ScienceDirect

Digital Investigation

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

Design tradeoffs for developing fragmented video carving tools

Eoghan Casey [a, *], Rikkert Zoun [b]

[a] Defense Cyber Crime Center (DC3), 911Elkridge Landing Rd., Linthicum, MD 21090, USA
[b] Netherlands Forensic Institute (NFI), Laan van Ypenburg 6, 2497 GB The Hague, The Netherlands

Keywords:
Data recovery
Digital forensics
File carving
File formats
DFRWS carving challenge
Fragmentation
Fragment reassembly

A B S T R A C T

When conducting a digital forensic examination, there is sometimes a need to salvage as much playable video as possible from available data sources. Although an ideal outcome might be to have all deleted and partially overwritten file fragments identified, reassembled, and repaired to provide playable videos, there are situations where this is not possible. In addition, there are complexities in real world datasets that can lead to false positives and false negatives. This paper captures practical lessons learned from extensive experiences in this problem space, and describes tradeoffs that developers must consider when creating file carving tools for salvaging and reassembling fragmented AVI, MPEG, and 3GP video files. Recommendations are provided for each tradeoff, concentrating on increasing the amount of playable video fragments that can be salvaged, with the potential for duplicate copies of some fragments being salvaged. Developers need to carefully consider how to handle the tradeoffs described in this paper when developing fragmented video carving tools. In addition, digital investigators need to consider the strengths and limitations of different fragmented video carving methods, and need to select those that are best suited to their given dataset. Another important outcome of this work is that the products of some carving methods may be playable in one video viewer but not others, making it necessary to view carved results using various methods, including storyboarding. This paper also includes discussion of current challenges and potential future work in fragmented file carving, with the aim of advancing research and development of automated methods for reassembling salvaged video fragments.

Introduction

File carving has become an important part of digital forensic examination, enabling practitioners to salvage deleted or concealed data. Until recently, carving was primarily based on headers (a.k.a. signatures) and, in some instances, limited validation of file format. Developments in file carving over the past few years have delved into specific file formats and associated data structures in an effort to piece together non-contiguous file fragments. These methods may be referred to as *semantic* file carving because they use knowledge of a specific file format to reconstitute pieces of a file.

In theory, digital video files are well-suited to semantic file carving methods because there are many characteristics in the format specifications that can be used to find and reassemble sub-pieces of an original file. However, in practice, there are complexities in the process that can lead to false positives and false negatives. In some instances, pieces of one video file are incorrectly associated with another video file or data are erroneously classified as video file fragment (false positive), and in other instances valid components of a video are missed (false negative).

* Corresponding author. Tel.: +1 202 670 2754.
E-mail address: eoghan@disclosedigital.com (R. Zoun).

http://dx.doi.org/10.1016/j.diin.2014.05.010
1742-2876/

Furthermore, although every effort is made to salvage videos in a form that can be replayed, some output files are too fragmented or corrupt to be viewed. To be practically useful in actual digital forensic investigations, a *playable* video is one for which some or all frames can be displayed by a widely available program such as FFmpeg [www.ffmpeg.org], MPlayer [www.mplayerhq.hu], VLC [www.videolan.org] or Windows Media Player.

Efforts to improve fragment carving and reassembly of video fragments in a particular context often result in the refined method obtaining less playable video under other conditions. Developers of carving tools that are widely used for actual casework must make tradeoffs between false positives and false negatives. It is also important for digital investigators to be aware of these tradeoffs to better understand the strengths and limitations of different fragmented video carving methods.

This paper presents practical lessons learned from extensive experience carving fragmented digital videos throughout the development of two widely used tools: Defraser and DC3Carver. This paper describes tradeoffs that developers must consider when creating semantic file carving tools for salvaging and reassembling fragmented AVI, MPEG, and 3GP video files. Recommendations are provided for each tradeoff, concentrating on increasing the amount of playable video fragments that can be salvaged, with the potential for duplicate copies of some fragments being salvaged. This paper concludes with current challenges and potential future work in automated video fragment reassembly.

Background

The most straightforward approach to salvaging deleted video files is to perform what is commonly called "header carving" by searching for a specific pattern of bytes (the file header) that is commonly found at the start of a file and to extract some preset amount of data after this header. This approach is effective when a file is stored in contiguous areas of the storage media, but it does not deal with fragmented files and, furthermore, it will not capture the full contents when the original file is larger than the preset amount of data that is extracted. For instance, if a video is 12 MB in size and only 10 MB of data is automatically extracted by the recovery tool, then 2 MB of video data will not be recovered. There has been continuous effort since 2007 to address the limitations of header carving in relation to video files.

Fragmented video carving methods

Some algorithms for automated reassembly of digital video fragments were developed as part of the DFRWS2007 fragmented file carving challenge [http://www.dfrws.org], resulting in prototype tools being developed (Cohen, 2007). Other approaches concentrated on finding and classifying file fragments, but did not address reassembly of video fragments (Garfinkel et al., 2010). More recent work has concentrated on recognizing and reassembling individual video frames (Na et al., 2014). The primary limitation of handling individual video frames is that it can result in a considerable number of carved fragments, from multiple videos, meeting the expected parameters.

One approach to improving video fragment reassembly is to make use of cluster boundaries in storage media (Lewis, 2011). Because files are generally saved on storage media per cluster, all data in a single cluster will belong to a single file, except for the last cluster of a file, which may also contain data from other file(s) in slack or uninitialized space. One of the challenges is to reliably detect clusters that contain video file data. Another challenge is to properly connect clusters that belong to the same fragmented file.

Netherlands Forensic Institute (NFI)

The NFI developed a tool named Defraser to help forensic examiners salvage deleted videos [http://defraser.sourceforge.net/]. Defraser parses through an input file for known characteristics of several video formats, including MPEG, 3GP and AVI. In addition to searching for headers that demark the beginning of a file, Defraser searches for characteristics of discrete segments within the file. In addition, this tool attempts to recognize common video compression formats (codecs), including H.264 and MPEG-1, -2 and -4 video. To reduce false positives, Defraser performs semantic checks such as the validity of header order, proper setting of bit flags, header parameter value constraints and continuity of incremental or cyclic parameter values, and where possible, complete decoding of video data.

All likely pieces of video are classified and then displayed in a graphical user interface to support forensic examination. Defraser enables forensic examiners to explore each potential video segment in great detail as per the file format specifications.

Although Defraser does not perform fully automated reassembly of fragmented videos, this tool enables digital investigators to reassemble and repair video fragments without in-depth knowledge of video formats. To support this process, Defraser provides a "Workpad" graphical user interface to reassemble and repair video fragments for replay and further analysis as discussed in Section Repairing Unassigned Fragments below.

Defense Cyber Crime Center (DC3)

Since 2007, DC3 has been developing DC3Carver (a.k.a. DCCI_StegCarver), a general purpose forensic carving tool that includes algorithms for carving numerous video types [http://www.dc3.mil/technical-solutions/tools]. Rather than using straightforward header carving approaches to salvage videos, DC3 developed carving methods that take advantage of the format and content of the individual video fragments that were found in the data set being carved. In addition to using file format characteristics to improve carving of physically contiguous videos, DC3 developed automated techniques to reconstruct and, when necessary, repair carved videos.

Realizing that different approaches are more effective in different situations, DC3 has developed two MPEG carving algorithms (which include the repair of some carved MPEG

files), both of which are integrated into DC3Carver; four AVI carving algorithms (which include some AVI repair features), two that are integrated into DC3Carver and two that are modularized into DC3_AVI-Carver and automatically called by DC3Carver; and three 3GP carving algorithms, one that is integrated into DC3Carver and two that are modularized into DC3_3GP-Carver and automatically called by DC3Carver.

Although the more recently developed algorithms are somewhat more sophisticated than earlier approaches, they are not considered replacements for the original algorithms. Testing on various datasets found instances when the less sophisticated algorithms produce more playable video than the more sophisticated algorithms. Therefore, the current version of DC3Carver contains all of the algorithms developed for automatically carving videos, including methods for carving both physically contiguous and fragmented videos.

Fragmented video carving tradeoffs

No single approach to salvaging and reassembling fragmented video files can address every case. Therefore, it is important for developers and digital investigators to understand the strengths and weaknesses of different approaches in order to use the most effective methods for a given case.

File system considerations

The question of what portion of a data set should be carved to produce optimal video carving results is sometimes difficult to answer. To produce the most playable video, the entire data set would have to be carved. However, the limitation of this approach is that it tends to produce a considerable number of duplicate files and false positives.

On the other hand, only carving unallocated space can be a problem because it can miss significant amounts of pertinent data, thus increasing the false negatives. For example, data contained in valid data length (VDL) slack, also referred to as uninitialized space, will be lost (Casey, 2011; Ferguson, 2008). Uninitialized space is quite common on NTFS used on CCTV recording systems, and on computers that are used to download video files via P2P applications because the download is prone to being interrupted because of the large file sizes that are being processed.

Furthermore, different forensic examination tools define unallocated space differently, which can lead to additional complications. Some forensic tools automatically attempt to recover deleted files that are still referenced in the file system, and then exclude the associated data from unallocated space. While this file system recovery process can reduce the size of the data set to be carved, and thus reduce the number of false positives, deleted file recovery can lead to a loss of pertinent data whenever the original file space has been overwritten.

Under most circumstances, carving a data set that includes traditional file slack, VDL slack, and all unallocated space (not excluding deleted files that are still referenced by the file system) will reduce the number of false positives and false negatives and thus produce the optimal number of playable carved videos.

Finding video fragments

When searching for video fragments, using criteria that are very specific can exclude valid fragments, whereas not being specific enough can result in many false positives.

Many video file formats have predictable data structures that can be searched for and assessed for validity. As an example, the Audio Video Interleave (AVI) file header has an abundance of information about the structure of a video summarized in Table 1, including the data needed to instruct a player how to render the AVI such as the encoding and a list of stream types (Microsoft AVI RIFF File Reference). The inherent structure of AVI files can be leveraged to scan input data for possible video fragments, and creating a catalog of their location and attributes (e.g., type, size) gives a map of available data and provides a foundation for reassembly operations.

Specifically, the AVI header list (hdrl) stores information about how many streams are defined within the header as well as information for playing the AVI. The AVI header (avih) identifies how many streams are defined within the header as well as information for playing the AVI. When dealing with fragmented AVIs, the total number of video frames becomes an important field for matching and validating the headers and indices. The video list (movi) stores the total size of all the AVI containers (a.k.a. chunks) followed by the individual chunks. The chunks stored in a

Table 1
AVI characteristics that are indexed for carving purposes.

Description	Hex values	Byte range
File header "RIFF"	\x52\x49\x46\x46	[0:3]
- Total size of AVI	-4 bytes (little endian)	- [4:7]
- "AVI"	- \x41\x56\x49\x20	- [8:b]
Header list "LIST"	\x4C\x49\x53\x54	[0:3]
- Size of header list	- 4 bytes (little endian)	- [4:7]
- "hdrl"	- \x68\x64\x72\x6C	- [8:b]
AVI header "avih"	\x61\x76\x69\x68	[0:3]
- Size of avi header	- 4 bytes (little endian)	- [4:7]
- Various flags		-
- # of video frames		- [18:1b]
LIST structures	\x4C\x49\x53\x54	[0:3]
- Size of list	- 4 bytes (little endian)	- [4:7]
- List type (movi, odml or stream)	- [movi, odml or stream]	- [8:b]
Chunk four character code ("db," "dc," "wb" or "tx")	[Hex of db, dc, wb or tx]	[0:3]
- Size of chunk	- 4 bytes (little endian)	- [4:7]
- Data	- [binary data]	- [8:size]
Index "idx1"	\x69\x64\x71\x31	[0:3]
- Size of index	- 4 bytes (little endian)	- [4:7]
- Index entries		- [8:size]
Index entry four character code ("db", "dc," "wb" or "tx")	[Hex of db, dc, wb or tx]	[0:3]
- Flags	- [flags]	- [4:7]
- Offset of chunk	- 4 bytes (little endian)	- [8:b]
- Size of chunk	- 4 bytes (little endian)	- [c:f]

movi container are pieces of data corresponding to an element of a stream type such as an audio segment or video frame. An AVI chunk contains a four character code and a size – the first two bytes indicate which stream in the header the chunk corresponds with and the next two bytes are the type (db = uncompressed video, dc = video, wb = audio, tx = text).

The optional AVI index (indx) contains the ordering information for frames, including an index marker, the size of the index and a series of index entries. This index information is stored as a four character code, a series of flags, the offset of the chunk, and the size of the corresponding chunk. Another item of note in AVI file format is that the four character code used in chunks is the same four character code used in the index. Since index entries are guaranteed to be sixteen bytes, the fragment cataloging process can look forward and behind to determine whether the code belongs to an index entry or to a chunk.

A direct comparison of carving results produced by different tools is difficult given the various methods implemented in each tool. To demonstrate, on a reformatted media card from a digital camera, Defraser detected 60 of 61 total AVI fragments. From these fragments, Scalpel salvaged eight playable videos (using 8 of 61 fragments), DC3Carver's contiguous carving method obtained fourteen playable videos (using 14 of 61 fragments), and DC3Carver's fragment reassembly method obtained 24 playable videos (using 51 of the total 61 fragments). These results indicate that semantic file carving methods can be more effective than header carving even when no fragment reassembly is attempted. Furthermore, the automated reconstruction of AVI fragments using DC3Carver resulted in playable video in many instances where contiguous carving failed to salvage playable video, demonstrating the potential benefit of semantic file carving. However, during this comparison, it was determined that Defraser excluded an AVI fragment, and that DC3Carver reassembled fragments in a different order, motivating further research and development. Finding these differences demonstrate the importance of comparing the results of multiple methods to salvage all available video fragments.

There are some sanity checks that need to be performed during the creation of a fragment catalog in order to reduce substantial numbers of false positive results. Some formats such as H.264 do not use clear, fixed identifiers, so creating a fragment catalog without carefully verifying the format will generate an enormous number of false hits. However, there are other circumstances in which a valid video can deviate from the file format specification. As a result, there is a risk that checking for strict adherence to a defined standard could result in playable video fragments being discarded. Therefore, it can be fruitful to attempt reassembly of video fragments even if a data structure does not completely conform to the file specification, but perhaps provide the user with some indication that the video deviates from its expected format specification. Furthermore, some components of video files may be corrupt, and a decision must be made whether to keep or discard fragments of questionable validity.

When creating such a fragment catalog, it is generally worthwhile to be over inclusive, whenever feasible. Any decisions to exclude specific items should be made with care in an effort to reduce the risk of excluding valid pieces of video files. In addition, it is more effective to search a data source for one type of video at a time, because performing simultaneous searches for multiple video types can interfere with each other, causing a fragment to be missed.

Measuring once, cutting twice

There may be more than one viable approach to reassembling video fragments, and it may not be feasible to determine automatically which approach will produce the most playable video.

For some digital video fragments, there may be sufficient information in data structures to determine that they fit together. For instance, when the actual size of a piece of digital video exactly (and uniquely) matches the expected size in the associated file header, this can provide a simple best-fit discriminator. Considerations when attempting to reassemble AVI fragments include the number of video frames in the index compared the number specified in the header's video stream list, and the potential for linking fragmented chunks by predicting a chunk's offset within a cluster from the known size of the preceding chunk.

Fragments of MPEG-2 and MP4 files can often be identified and reassembled with a high degree of precision because the file structure is quite strict and contains information that can be used to match related fragments. In addition, the MPEG-2 Systems (ISO 13818 part 1) format and MPEG-4/H.263 (ISO 14496 part 2) and H.264 formats store video frame parameters that can be expected to have a predictable progression, such as time codes or other (possibly cyclic) counters. For example, Table 2 is an example of an MPEG-2 Systems video fragment, showing Pack Headers with increasing SystemClock Reference values, as well as certain PesPackets showing Decoding and Presentation time stamps that can be used to link different fragments that are found when carving.

The MP4 file format (ISO 14496 part 1) and Apple's QuickTime format (ISO/IEC 14496-12, Apple Inc.) both have three main components:

- *container header*: a header that specifies the type of data in the container.
- *moov*: video track container which stores all of the metadata for a video, including frame information and track listings.
- *mdat*: media data container which stores the actual video/audio data.

In addition to this overarching structure, individual components (such as separate video frame data) have temporal sequencing information. For instance, Table 3 shows an MP4 file fragment (ISO 14496 part 14) containing MPEG-4 Video (ISO 14496 part 2) from the DFRWS2007 Forensic Challenge with TimeIncrement values that can be used to link different fragments that are found when carving. When an MP4 file fragment contains H.264 encoded video, the FrameNumber values can be used to link salvaged fragments.

Table 2
MPEG-2 Systems fragment (ISO 13818 part 1) containing MPEG-2 Video (ISO 13818 part 2) from DFRWS2007 Forensic Challenge showing TimeIncrement values.

Offset	Size (bytes)	Type	Attribute
274370048	14	PackHeader	SystemClockReference = 0
274370062	18	SystemHeader	
274370080	2016	PesPacket	PresentationTimeStamp = 0, DecodingTimeStamp = "8589930992"
274370103	12	SequenceHeader	
274370125	8	GroupOfPicturesHeader	TimeCode = "00:00:00-00
…			
274372096	14	PackHeader	SystemClockReference = 900
…			
274372110	2034	PesPacket	PresentationTimeStamp = 0
274374144	14	PackHeader	SystemClockReference = 1800
…			
274416688	12	SequenceHeader	
274416710	8	GroupOfPicturesHeader	TimeCode = "00:00:00-10

This type of information creates a very exact picture of the data that is stored, which generally enables accurate reassembly of salvaged fragments whether the file is contiguous or fragmented. The structure and metadata also make it feasible to play partial or corrupt videos.

MPEG files are often large and fragmented and, therefore, significantly more video data can be recovered using semantic file carving approaches when compared with header-based file carving tools. The effectiveness of MPEG fragment reassembly can be demonstrated with the DFRWS2007 carving challenge, using DC3Carver to successfully carve out every video in the dataset in its entirety. The salvaged video includes a three minute MPEG-2 video that is broken into more than 30 fragments that can be reassembled into a playable video with continuous sound and synced audio and video.

In other situations, reassembly is less straightforward. For instance, when the fragments from one video file are intermingled with another video file, there is a chance that attempting to carve each of them out separately may render one or both of the videos unplayable. But there is also a chance that separating them will result in both of them being playable. Furthermore, in some situations, more playable video can be salvaged using simple header carving, without attempting to reassemble file fragments. Therefore, it is sometimes advisable to salvage and reassemble video fragments using multiple approaches even though it results in duplicate data. Subsequent validation of file format and playability can be performed on the results to eliminate unplayable video (see Section Determining Whether Videos Are Playable for further discussion regarding the strengths and limitations of validation).

As an example, Fig. 1 depicts video files in 3GP format that have been partly overwritten by other data. As background, the 3GP format follows the MP4/Quicktime file specification, starting with a distinctive file type header ("ftyp") that stores information about compatibility (types listed at http://ftyps.com/). When such files contain video

Table 3
MP4 fragment containing MPEG-4 Video from DFRWS2007 Forensic Challenge showing TimeIncrement values.

Offset	Size (bytes)	Type	Attribute	Comment
228514304	20	ftyp	MajorBrand = "isom"	CompatibleBrands = "mp41"
228514324	2,445,504	mdat		
228514332	18,457	Vop	TimeIncrement = 0	CodingType = "I_VOP"
228533351	3976	Vop	TimeIncrement = 1	CodingType = "P_VOP"
228537740	2248	Vop	TimeIncrement = 2	CodingType = "P_VOP"
228540402	5001	Vop	TimeIncrement = 3	CodingType = "P_VOP"
228545829	6195	Vop	TimeIncrement = 4	CodingType = "P_VOP"
228552470	4580	Vop	TimeIncrement = 5	CodingType = "P_VOP"
228557499	6300	Vop	TimeIncrement = 6	CodingType = "P_VOP"
228564245	5507	Vop	TimeIncrement = 7	CodingType = "P_VOP"
228570071	5339	Vop	TimeIncrement = 8	CodingType = "P_VOP"
228575902	5180	Vop	TimeIncrement = 9	CodingType = "P_VOP"
228581586	4525	Vop	TimeIncrement = 10	CodingType = "P_VOP"
228586643	5638	Vop	TimeIncrement = 11	CodingType = "P_VOP"
228592843	18,946	Vop	TimeIncrement = 12	CodingType = "I_VOP"
228612368	4544	Vop	TimeIncrement = 13	CodingType = "P_VOP"
228617492	2789	Vop	TimeIncrement = 14	CodingType = "P_VOP"
…				
230959828	16,635	moov		
230959944	4597	trak	TrackID = 1	Video
230964541	11,792	trak	TrackID = 2	

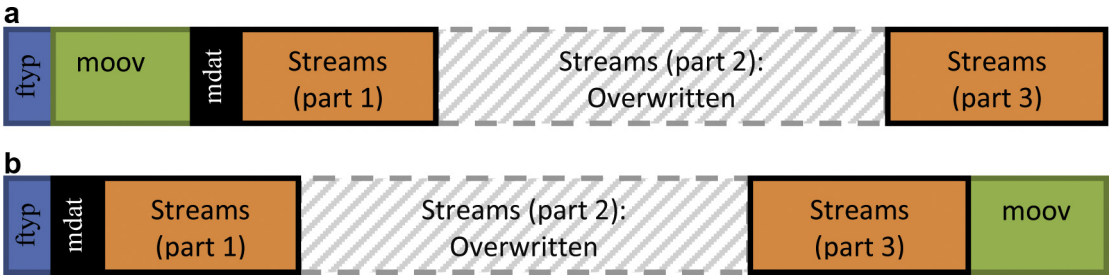


Fig. 1. a and b: Fragmented 3GP video files.

in a format that does not store frame sequence information that could be useful for reassembling fragments, it is necessary to try other methods to match and reassemble such fragments into a playable 3GP video. Although the ftyp header must be at the beginning of a 3GP file, the moov container can be placed before or after the mdat container. Essentially, the moov container provides an index detailing where each *atom* of data containing video frames and audio samples are located within a 3GP file. Each atom in a 3GP file begins with fields that specify the size and type/format of the atom, followed by any other data stored in that atom.

In the top example, Fig. 1a, only carving to the end of the fragment labeled "part 1" would result in an invalid video because references in the file header would point outside of the salvaged file. In addition, the fragment labeled "part 3" would not be included in the salvaged video, and may be missed entirely. In the bottom example, Fig. 1b, only carving to the end of the fragment labeled "part 1" would cause a crucial component at the end of the file to be separated from the start of the file, thus missing metadata required for demuxing part 1. Therefore, it is most effective to carve the entire space in both scenarios to produce the most playable video. Playback may falter at the end of part 1 because the data in the overwritten part 2 may confuse the decoder, but playing might resume if the decoder is able to continue processing (i.e., does not abort or hang when it encounters the corrupted portion) until it reaches the streams that make up part 3. In both instances, a remaining challenge is to determine whether fragments of other videos found in part 2 can be made playable as discussed in Section Repairing Unassigned Fragments.

A major challenge to reassembling fragmented 3GP files is that the bulk of video data is within the mdat section, which usually consists of video data conforming to particular video encoding standards (most commonly MPEG-4 video, H.263 or H.264) interleaved with audio data conforming to particular audio encoding standards (most commonly AMR or MPEG-4 audio). Any fragmentation of the mdat section will cause the link between the moov and the mdat sections to be lost.

In 2011, DC3 developed a refined approach to carving physically contiguous 3GP file fragments that uses file format characteristics, and a second carving method to extract, order, and reassemble 3GP fragments. In addition, to address this issue, Defraser performs additional parsing of file fragments using codec detectors. First, for videos without an associated moov component, Defraser truncates the mdat to 8 bytes (indicating no payload) and associates that atom with the ftyp, and then uses codec detectors to handle any video fragments that may still be in the mdat. Specifically, Defraser performs a cursory scan for MPEG-4/H.263/H.264 start codes at sample locations to determine where the first invalid sample occurs. The resulting truncated streams are then scanned with the codec detectors, including the required headers that are in the moov, so the frames should be decodable and viewable.

Another approach to carving 3GP videos is to take an mdat even if it does not have an ftyp header, and combine it with any matching moov. This can result in a large amount of false positive results, but many may be eliminated by file validation. In addition, Defraser forensic logs can be generated when saving a fragment and used to determine MPEG-4 or H.264 frame positions and sizes, and match those with sample tables in any detected moov component. As another example, when multiple partial copies of the same video are found, one fragment could fit in all instances. In such circumstances, it may be necessary to reuse a single fragment to reconstruct multiple partial copies of the video in order to obtain the most playable video.

Repairing unassigned fragments

It may be possible to repair corruption in an existing header or to salvage remaining fragments either by reconstructing a valid container around a fragment, or by grafting an appropriately formatted header onto the separate video and audio streams.

Any fragments that could not be associated with a valid video header may still be viable for replay after a "reference header" is grafted onto them. For instance, when a header is missing, some MP4 videos will play when a generic 3GP header is grafted on to the fragment, provided the original moov header that contains the sample tables (stco, stsz, stsc) is still intact. For AVI videos, grafting on various common header types (e.g., cvid, mjpg, mlre, xvid) can make video fragments playable. This technique can be useful for grafting reference headers for H.264 or MPEG-4 video. The latest version of Defraser (1.4.1) enables users to create an internal database of reference headers from available videos, and can automatically graft these reference headers onto frames that lack headers. Any frames that are decoded using a reference header will show up in the GUI including that header, and can thus be saved in playable form without the need for manual repair.

As an example, Fig. 2 shows a reference header database created using Defraser, and Fig. 3 shows this tool being used to repair an MPEG-1 Video file by grafting a reference header onto the start of the video stream. The SequenceHeader at the top of Fig. 3 comes from a reference video, the GroupOfPictureHeaders and PictureHeaders are from the evidence, and were unplayable before the automated repair.

It can also be effective to take the header from a playable video file that was carved from the same data source, and graft this reference header onto salvage video fragments. For example, using Defraser, a header from a playable video in the dataset can be dragged into a Workpad within Defraser, and then the unplayable video file fragment elements can be dragged into the Workpad, thereby creating a valid video file. After selecting all headers in the Workpad and saving it, the resulting file is playable. Optionally, a forensic integrity log can be saved along with the video file, in which the source of all video elements is documented.

Another approach to repairing a video fragment is to reconstruct a container around a fragment, based on detected video frame and audio sample locations and (if available) a reference video file with similar parameters. This technique requires a method or tool such as Defraser to properly detect frame offsets and sizes that can be used to reconstruct a header with the correct locations and sizes in order to play the video with its original synchronization.

Whatever method is used, for forensic purposes, any such repair of fragments needs to be differentiated in some manner to convey to the user that a header that was not part of the original evidence has been added to the file.

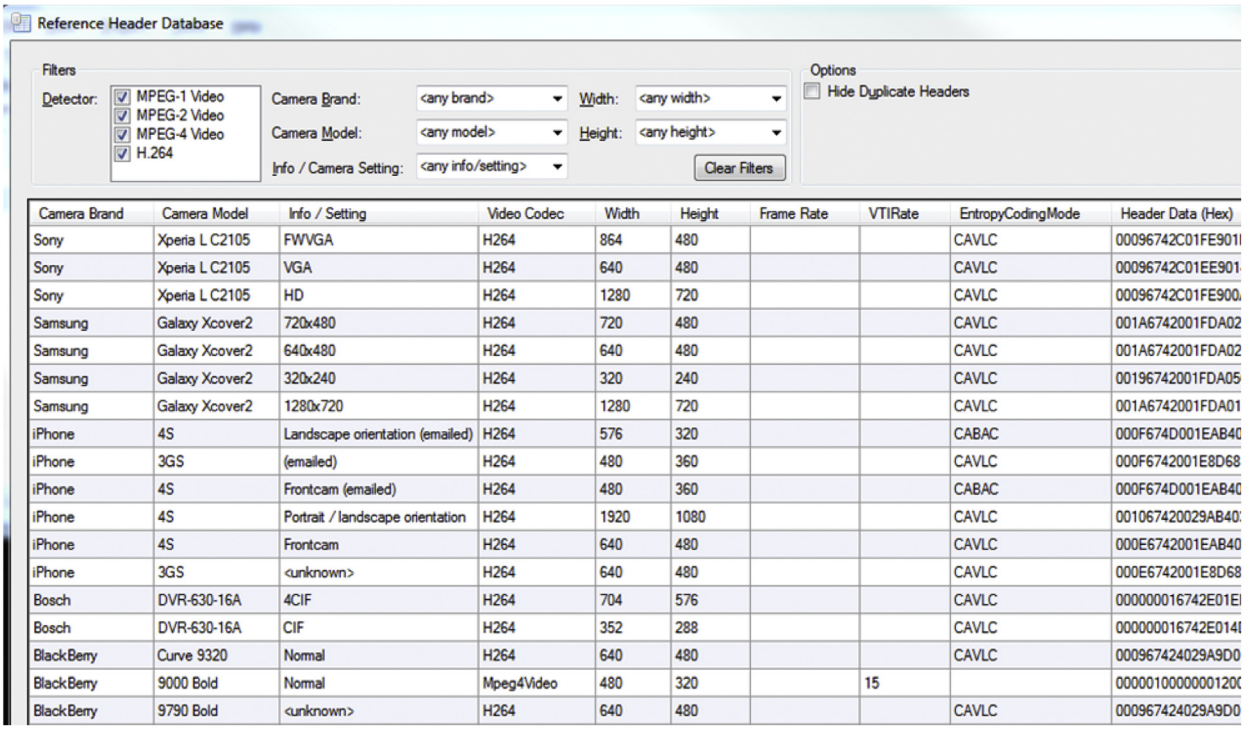
Determining whether videos are playable

A further consideration for any file carving of videos is that salvaged files might be playable using some tools but not others, or display different content when played with different tools. For example, some reassembled videos might only play in a particular version of FFmpeg, or in a different media player. This may for instance be caused by implementation-specific boundary conditions, such as the requiring of a minimum number of frames or the presence of at least one key frame before decoding is at all attempted by the software. In some instances, a salvaged video may contain portions of two unrelated videos as shown in Fig. 4. As a result, the salvaged file may play one series of frames when viewed using one player, and may play a different series of frames when viewed using a different player, making it difficult to determine whether a reassembled video can be played in part or in full.

To address this issue, it is necessary to use multiple methods for determine whether a reassembled video has a valid format and can be played, including codec validation using a tool such as FFmpeg. The DC3 VideoValidator uses MPlayer and multiple versions of FFmpeg to extract potential frames from video files.

Conclusions and future work

The structured format of digital video makes it well-suited for semantic file carving, which can find and reconstruct fragments into playable files that more basic carving methods miss.



Camera Brand	Camera Model	Info / Setting	Video Codec	Width	Height	Frame Rate	VTIRate	EntropyCodingMode	Header Data (Hex)
Sony	Xperia L C2105	FWVGA	H264	864	480			CAVLC	00096742C01FE901
Sony	Xperia L C2105	VGA	H264	640	480			CAVLC	00096742C01EE901
Sony	Xperia L C2105	HD	H264	1280	720			CAVLC	00096742C01FE900
Samsung	Galaxy Xcover2	720x480	H264	720	480			CAVLC	001A6742001FDA02
Samsung	Galaxy Xcover2	640x480	H264	640	480			CAVLC	001A6742001FDA02
Samsung	Galaxy Xcover2	320x240	H264	320	240			CAVLC	00196742001FDA05
Samsung	Galaxy Xcover2	1280x720	H264	1280	720			CAVLC	001A6742001FDA01
iPhone	4S	Landscape orientation (emailed)	H264	576	320			CABAC	000F674D001EAB40
iPhone	3GS	(emailed)	H264	480	360			CAVLC	000F6742001E8D68
iPhone	4S	Frontcam (emailed)	H264	480	360			CABAC	000F674D001EAB40
iPhone	4S	Portrait / landscape orientation	H264	1920	1080			CAVLC	001067420029AB40
iPhone	4S	Frontcam	H264	640	480			CAVLC	000E6742001EAB40
iPhone	3GS	<unknown>	H264	640	480			CAVLC	000E6742001E8D68
Bosch	DVR-630-16A	4CIF	H264	704	576			CAVLC	000000016742E01E
Bosch	DVR-630-16A	CIF	H264	352	288			CAVLC	000000016742E014
BlackBerry	Curve 9320	Normal	H264	640	480			CAVLC	000967424029A9D0
BlackBerry	9000 Bold	Normal	Mpeg4Video	480	320		15		0000010000000120
BlackBerry	9790 Bold	<unknown>	H264	640	480			CAVLC	000967424029A9D0

Fig. 2. Example of a Defraser reference header database.

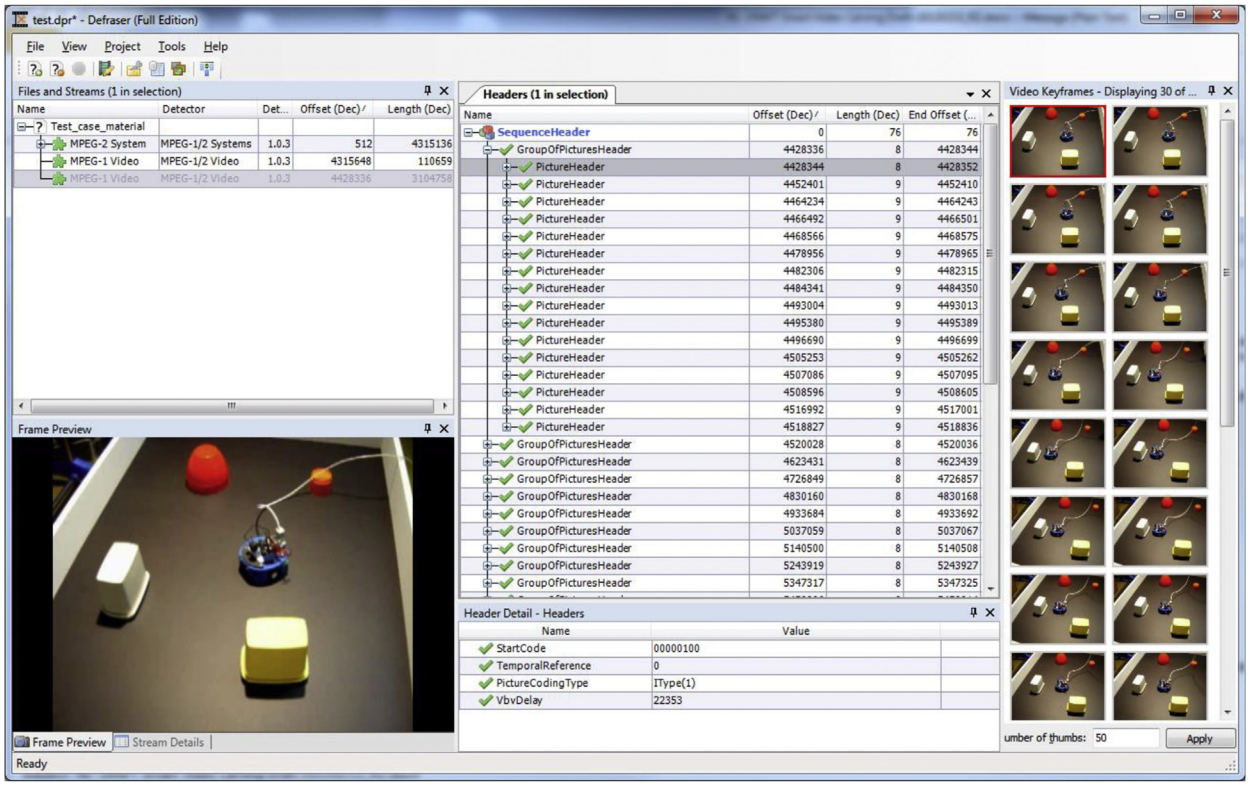


Fig. 3. Automated MPEG Video fragment repair using MPEG Video reference header from database.

From a practitioner perspective, the most playable video can be salvaged from deleted areas of storage media by extracting all unallocated and slack space (including VDL slack), running multiple tools that implement different methods (including header repair), and then reviewing the results using various video players and a tool such as DC3 VideoValidator.

Given the complexities encountered in real world datasets, no single approach to salvaging fragmented video files and rendering them playable will be most effective in all cases. Additional research and development is needed to create new fragment reassembly methods that are more effective in particular circumstances. At the same time, there is a risk that more refined methods will obtain less playable video under other conditions.

Therefore, developers of fragmented video carving tools must carefully consider how to handle the tradeoffs to improve results in certain situations. In addition, digital investigators need to consider the strengths and limitations of different fragmented video carving methods, and need to select methods that are best suited to their given dataset. It is generally advisable to use multiple methods in order to obtain more playable video. Digital investigators must also be aware that output from some carving methods may be playable in one video viewer but not others, making it necessary to view carved results using various methods, including storyboarding.

Semantic video carving could also be improved by including popular video encoding standards, such as MPEG-4 Video and H.264. If the location of individual video frames can be detected directly within a video container using the relevant specifications, one would not be so dependent on availability of indexes from container formats; and the video frame locations could then be determined more locally. Such location information could be used to generate an appropriate container video file index for a partial file, as a step in the reassembly of a playable video file. In such cases, the availability of a reference video that was recorded with the same settings is very helpful. Additionally, decoding encoded video streams may provide more clues as to the original ordering of any fragmented video file fragments that were found by carving.

For example, future work related to salvaging fragmented 3GP video files could attempt to reconstruct a fragmented mdat so that all available frames could be reproduced instead of partial video reproduction. Generally, in a large video, the mdat accounts for a huge percent of the file size so it is likely that any fragmentation would occur within this part a majority of the time. Ultimately, methods of matching fragment location patterns with fragment location indexes require some robustness in order to handle potential false hits in frame detection. This is less of an issue for AVI, but it can be more problematic with formats that use less unique identifiers such as H.264 or MPEG-4 audio.

A closely related area for future work is improved methods for repairing partially salvaged video to produce playable video and/or audio content, either separately or

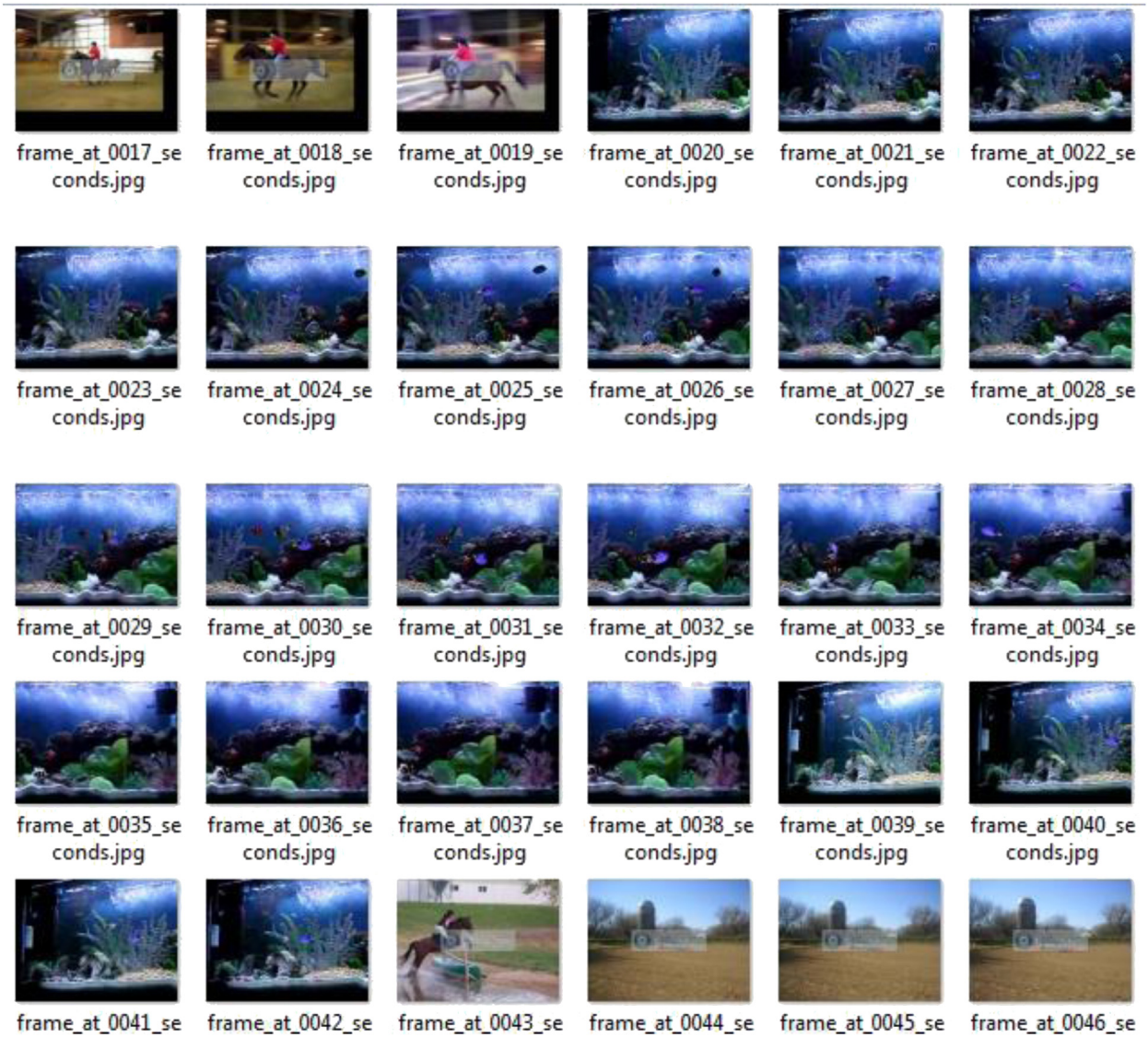


Fig. 4. Storyboard of reassembled MPEG video fragments from the DFRWS2007 Forensic Challenge database, with frames from two videos interleaved, extracted using DC3 VideoValidator with ffmpeg10 [http://video-validator.sourceforge.net].

with the original synchronization intact. In some situations, suitable headers for grafting onto video frames may be found in other areas of the data source, because videos may have been recorded with similar settings. Alternatively, if the originating camera is known (e.g., when analyzing a memory card from a mobile phone or camera), one could record a reference video, from which to use the headers.

A complication to decoding individual frames is that a certain decoder state is required, which is gained by decoding a particular MPEG header. For instance, state information is contained in H.264 Sequence Parameter Set and the Picture Parameter Set headers, and in the Sequence Header and the Sequence Extension of MPEG-2 Video, and crucial playback information such as image resolution is contained in the MPEG-4 Video Object Layer. Such a header may for instance be found in other parts of the data source, or borrowed from a proper reference video file, or constructed manually using the format specification and some educated guessing. Defraser 1.4.1 supports using reference headers, from which decoder state is generated when needed. This approach allows fragments that have no header to still be decodable, if the right reference header is set. In casework where recordings by a known mobile camera are sought, one would typically use a reference movie from such a camera to determine the specific file format characteristics.

Acknowledgements

Thanks to Mark Hirsh for his continued work and contributions, and to Jason Agurkis, Keith Bertolino, Matt Nolan, and other contributors to DC3Carver. In addition,

thanks to Zeno Geradts for his efforts and support related to this work.

References

Apple Inc. Apple QuickTime file format specification, http://developer.apple.com/library/mac/#documentation/QuickTime/QTFF/QTFFChap1/qtff1.html#//apple_ref/doc/uid/TP40000939-CH203-SW1.

Casey E. Digital evidence and computer crime. 3rd ed. Elsevier, Academic Press; 2011.

Cohen M. Advanced carving techniques. Digit Investig 2007;4(3–4). 1419.

Ferguson D. Redefining file slack in Microsoft NTFS. J Digit Forensic Pract 2008;2(3).

Garfinkel S, Nelson A, White D, Roussev V. Using purpose-built functions and block hashes to enable small block and sub-file forensics. In: DFRWS2010 Conference Proceedings, Supplemental Issue of Digital Investigation, vol. 7; 2010. Available online at: http://www.dfrws.org/2010/proceedings/2010-302.pdf; 2010.

Lewis AB. Reconstructing compressed photo and video data (PhD thesis). University of Cambridge Computer Laboratory; June 2011. Technical Report UCAM-CL-TR-813, Chapter 5: Reconstruction of fragmented compressed data, http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-813.html.

Microsoft AVI RIFF File Reference, http://msdn.microsoft.com/en-us/library/ms779636.aspx.

Na G, Shim S, Moon K, Kong SG, Kim E, Lee J. Frame-based recovery of corrupted video files using video codec specifications. IEEE Trans Image Process 2014;23(2).