

# Forensic analysis of the filesystem

## Dahua DHFS 4.1

Dane Wullen

Email: dane.wullen@gmail.com

**Abstract**—File systems are the foundation for managing data on digital media. They serve to structure and efficiently store files such as images, documents, and videos. Many of these file systems, e.g., FAT, NTFS, HFS+, etc., from well-known operating systems and manufacturers, have been analyzed in numerous studies and can be interpreted by most forensic programs. Besides these well-known file systems, there are lesser-known, mostly proprietary file systems that are increasingly coming into focus in digital forensics.

Among other, many manufacturers of network video recorders implement these unknown file systems to efficiently store video data. For digital forensics, this video data can be a crucial key to solving and investigating crimes..

This work deals with the DHFS 4.1 file system from the manufacturer DAHUA. Based on file system analysis, the structure and file storage management are explained and how methods of digital forensics can be applied to locate video files is demonstrated. Based on these findings a X-Tension for the program X-Ways Forensics is developed to interpret and analyze digital storage media that implement this file system.

**Keywords**—DHFS 4.1, DAHUA, DVR, NVR, file system

### I. INTRODUCTION

Digital forensics is a relatively new discipline in the investigation of crimes. Beginning in the 1980s, digital forensics experienced a leap forward from the year 2000 onwards, as more and more people gained access to computers, mobile phones and the internet.[1]

In addition to traditional analog traces and evidence used to solve crimes, digital evidence such as mobile phones, computers, and smartphones is becoming increasingly important. Those evidence contains electronic traces such as images, documents, emails, and chat histories, which can be relevant to solving crimes.

Digital video surveillance systems (e.g. digital video recorders, DVRs) play a crucial role in preventing and solving crimes. Depending on their capacity, these systems record the areas captured by the cameras around the clock and can be highly valuable in law enforcement.

However, many of these systems do not use file systems such as FAT, exFAT, or NTFS, which are well-known and analyzed in digital forensics. Instead, the manufacturers of these video surveillance systems implement their own proprietary file systems to manage the video data.

In order to read and interpret the data on these file systems, the file systems must be analyzed and the structure and management of the storage of data has to investigated as this is the only way to extract the data.

The aim of this work is to analyze the proprietary file system DHFS 4.1 from the manufacturer DAHUA. Based on various digital storage media, the file system is analyzed section by section and the underlying video data is extracted.

The results of this analysis will be used to development an extension for the program X-Ways Forensics, a so-called X-Tension. This X-Tension will then allow the program to read, interpret, and analyze digital storage media or images, making existing and partially deleted video files visible and evaluable for investigators.

### II. RELATED WORK

The foundation for file system analysis is Brian Carrier's work "File System Forensic Analysis". Carrier describes in this book the precise analytical steps necessary to examine a file system and classify its components into specific categories. [2]

In the field of video surveillance systems, there are several related studies that deal with the analysis of proprietary file systems or the video files of these systems. The most relevant work is from Jaehyeok Han et al., in which the file system of the video surveillance system manufacturer HIKVISION is analyzed. Han et al. describe the structure of the file system and the video files that are stored within it.. [3]

Another related work by Lee Tobin et al. describes the general analysis of unknown file systems from video surveillance systems. This work examines various methods for the manual analysis of file systems, including the identification of file system offsets and repeating structures, as well as the linking of this data and information. [4]

In another work, Von Dongen describes the analysis of a file system and the proprietary file formats of a video recorder manufactured by Samsung. [5]

Only a small number of publications specifically about the manufacturer DAHUA exist. Evangelos Dragonas et al. describe in their work the analysis of log files from a DAHUA

DVR. These log files also contain information essential for digital forensics regarding the configuration and user behavior and can provide investigators with insight into the system. [6] However, these are not the log files that are analyzed in this file system.

### III. STATE OF THE ART

In addition to current work and publications dealing with file system analysis, especially among manufacturers of video surveillance systems, a look is taken at the current state of the art, specifically at programs capable of analyzing these file systems..

One of the best-known programs is Witness, formerly DVR-Examiner, from Magnet Forensics. According to the manufacturer, this program is capable of interpreting and analyzing a wide variety of file systems from video surveillance systems. User reports indicate that video files from DAHUA's proprietary file system can also be viewed with Witness. It searches for and recovers both existing and deleted video files. [7]

DiskInternals also offers a program for viewing and saving videos from various file systems used by DVR manufacturers, including HIKVISION and DAHUA. [8] A trial version is available free of charge; however, the purchase of a license is required for continued use.

In addition to paid programs, there are also several open-source solutions available that can recognize and extract data from the proprietary DHFS 4.1 file system..

Galileu Bastista developed a Python program that can display and extract stored videos from a DAHUA video surveillance system's hard drive image. The program can also search for deleted files based on their signatures. The program code partially describes the file system structure and is available on GitHub. [9]

Another project on GitHub by user DmytroMoisiuk, also a Python program, searches for video data based on a signature and attempts to link it. However, this program works purely as a file carver, as it does not consider the structure of the file system. [10]

The manufacturer DAHUA offers the program SmartPlayer. [11] The program can display existing video files on a hard drive using the DHFS 4.1 file system and save them in the proprietary file format (.dav) on a computer. However, it does not recover deleted files.

### IV. FILE SYSTEM ANALYSIS

This section describes the structure of the DAHUA DHFS 4.1 file system. It uses Carrier's methodology to divide the file

system into its various components. Similar to Lee Tobin et al., most elements in the file system structure were identified through testing and manual sector scanning. Recurring patterns, areas, and values were observed to reveal the overall structure.

A classic file system typically consists of several parts that describe the different areas. There is a boot sector, a main directory, and areas where the data and metadata are stored. To determine where a file system begins, an operating system reads a partition table, such as the Master Boot Record (MBR) or the GUID Partition Table (GPT), which contains the entry point and size of the file system.

Carrier uses five categories for analyzing a file system: file system, file content, metadata, file name, and application data. Each of these categories contains data that fulfills various tasks, such as locating important sections, determining the entry point of data, etc. The following describes each category using the DHFS 4.1 file system as an example.

#### A. File system category

Data about a file system are essential data that describes the structure of the file system. For example, it describes the size of the file system, the size of an allocation unit (e.g., cluster or inode), or the addresses of the data or metadata sections.

Essential data is distinguished from non-essential data in that it must be „trusted“. If the data is corrupt or missing, an analysis of the file system cannot be carried out without further ado.

To correctly read the values in the following sections, the byte order, also called endianness, must first be determined. In DHFS 4.1, text and signatures are read in big-endian format, and numeric values in little-endian format.

The first sector of a DHFS 4.1 formatted data storage device begins with the signature 0x44 48 46 53 34 2E 31, which stands for the ASCII representation „DHFS4.1“. This signature can be used to determine whether DHFS 4.1 is present, see figure.1.

DHFS 4.1 has two sections that contain data about the file system: a partition table and a boot sector for each partition. Normally, the partition table of a storage device is not considered file system data, as it only contains the entry point to the file system.

However, in DHFS 4.1, the partition table is unique in that it is located within the context of the first partition. Therefore, it is considered part of the file system data.

1) *Partition table:* The partition table is located in sector 30 from the beginning of the disk. The first 32 bytes are unknown and are not needed for the analysis.

000000000000	44 48 46 53 34 2E 31 00 00 00 00 00 00 00 00 00	DHFS4.1
000000000016	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000032	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000064	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000096	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000112	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000144	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000160	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000176	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000192	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000208	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000224	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000240	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000256	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000272	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000288	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000304	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000320	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000336	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000352	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000368	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000384	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000400	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000416	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000432	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000448	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000464	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000480	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
000000000496	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Figure 1. First sector of DHFS 4.1

Offset	Description	Color
0x08 - 0x0B	Offset to boot sector	Red
0x24 - 0x2B	Offset to partition	Blue
0x2C - 0x2F	Length of partition	Yellow
0x134 - 0x137	Magic number	Green

Figure 2. Partition table

Table I  
BYTE OFFSETS PARTITION TABLE

Offset	Description	Color
0x08 - 0x0B	Offset to boot sector	Red
0x24 - 0x2B	Offset to partition	Blue
0x2C - 0x2F	Length of partition	Yellow
0x134 - 0x137	Magic number	Green

This is followed by 32 bytes describing the individual partitions of the file system. The most important information includes the address of the boot sector, the length of the partition, and its entry point; see table I and figure 2. All values are to be interpreted in sectors, where one sector contains 512 bytes.

The end of the partition table is marked with the hexadecimal value 0xAA55AA55, also known as the magic number. No further partitions follow.

It can be assumed that a partition has a maximum size of  $2^{32}$  sectors. Therefore, a maximum of 4.294.967.296 sectors \* 512 bytes, or 2 TB, can be used for a partition. The examined disk images generally had 4 partitions, so theoretically, in this case, disks with a size of 8 TB could be used for this file system.

2) *Boot sector of a partition:* The boot sector, which is usually located in sector 34 from the start of the partition, contains essential information about the entry points to the data and metadata categories as well as the sizes of the allocation units, see figure. 3.

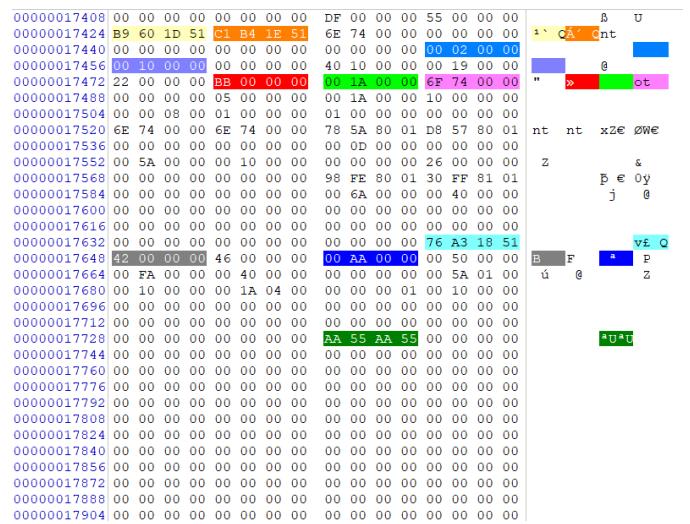


Figure 3. Boot sector of the first partition

Each boot sector has several timestamps, including the supposed creation timestamp of the partition or log file and two timestamps that limit the recording period for the respective partition.

The size of a sector and the size of an allocation unit are defined in the boot sector. Following the conventions of NTFS and FAT, these allocation units are also referred to as clusters in this work. It was found that the sizes were consistently identical in the examined disk images: 512 bytes per sector and 4096 sectors per cluster. Further values can be found in table II.

Table II  
BYTE OFFSETS OF A BOOT SECTOR

Offset	Description	Color
0x10 - 0x13	Start of record	Yellow
0x14 - 0x17	End of record	Orange
0x2C - 0x2F	Byte per sektor	Blue
0x30 - 0x33	Sektor per cluster	Purple
0x44 - 0x47	Offset to descriptor table	Red
0x48 - 0x4B	Offset to video data area	Green
0x4C - 0x4F	Entries in descriptor table	Pink
0xEC - 0xEF	Creation date Log	Cyan
0xF0 - 0xF3	Offset to spare boot sector	Grey
0xF8 - 0xFB	Offset to log files	Dark Blue
0x148 - 0x14B	Magic number	Dark Green

The entry point to the video data area and the start of the descriptor table, which describes the individual video files, is generally the same for all boot sectors, but can vary from disk to disk. The data area in figure 3 begins at sector 6656, the descriptor table at sector 187. The size of the descriptor table varies depending on its contents, as the value represents the number of 32-byte values used.

In addition to the offsets to the data and metadata categories, each partition has a offset to its own log file. The offset to this log file is, for example, sector 43520 and is described in the application data category.

As in the partition table, the marker 0xAA55AA55 marks the end of the boot sector.

In addition to the boot sector in sector 34, each partition also has a spare boot sector, the entry point of which is defined at offset 0xF0 - 0xF4 of the respective partition. This spare boot sector is a 1:1 copy of the first boot sector and can be read in case of damage or corruption of the first boot sector.

The boot sectors of DHFS 4.1 have certain similarities to the boot sector of an NTFS partition, as a similar end marker is used and a copy of the boot sector exists. The sector and cluster sizes are also defined.

3) *Descriptor table:* The descriptor table is one of the most important data structures in DHFS 4.1. The function of this data structure also relates to the categories of data and metadata, which will be discussed later.

Basically, the descriptor table is a continuous, doubly linked list starting with index or ID 0. Each list entry is 32 bytes in size and contains the allocation status of a DHFS 4.1 cluster. An entry or descriptor is either free (0xFE), a main video descriptor (0x01), or a video fragment descriptor (0x02), see figures 4 to 6. No other allocation types are known.

00000095744	FE	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	b
00000095760	00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	b
00000095776	FE	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	b
00000095792	00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	b

Figure 4. Empty descriptor

00000228864	01 23 08 08	B9 60 1D 51	00 70 1D 51 42 10 00 00	# H 1` Q E CB
00000228880	00 0E 00 00	00 00 00 00	40 10 00 00 00 00 00 00	0
00000228896	01 21 0A 03	BB 60 1D 51	00 70 1D 51 44 10 00 00	! » Q p QD
00000228912	E0 0F 00 00	00 00 00 00	41 10 00 00 00 00 00 00	A

Figure 5. Main video descriptor

00000229376	02 23 06 00	DC 60 1D 51	E1 60 1D 51 52 10 00 00	! U Q CR
00000229392	00 00 00 00	4D 10 00 00	41 10 00 00 00 00 00 00	M A
00000229408	02 23 08 00	DF 60 1D 51	E3 60 1D 51 53 10 00 00	# B Q S
00000229424	00 00 00 00	4F 10 00 00	40 10 00 00 00 00 00 00	O @

Figure 6. Video fragment descriptor

The main video descriptor differs from the video fragment descriptor in that it serves as a kind of entry point for the cluster chain. It describes the total length of the video, the selected channel (i.e., the selected camera), the number of fragments, and the ID, thus the (partition-wide) unique name of the video. It also describes the size of the last fragment.

The recording period for a main video descriptor is one hour, except when a video recording is interrupted. DHFS 4.1 uses a different definition for determining the time than the UNIX timestamp, which is described in the metadata category.

The video fragment descriptor describes a single fragment that is assigned to a main video descriptor. It contains information about the fragments that precede or follow it, a recording time period, and its relation to the main video descriptor.

The data structures of the main video descriptors and video fragment descriptors differ in a few points. The main video descriptor stores the number of existing video fragments and the size of the last video fragment, and has no predecessor. The video fragment descriptor stores its predecessor and the fragment's internal number to ensure correct sequencing. These differences are illustrated in the tables III and IV.

Table III  
BYTE OFFSETS MAIN VIDEO DESCRIPTOR

Offset	Description	Farbe
0x00 - 0x00	Status	Yellow
0x01 - 0x01	Channel	Orange
0x02 - 0x03	Number of video fragments	Green
0x04 - 0x07	Start of record	Cyan
0x08 - 0x0B	End of record	Blue
0x0C - 0x0F	Next descriptor ID	Pink
0x10 - 0x11	Size of last video segment	Dark Green
0x14 - 0x17	Null	Purple
0x18 - 0x1B	Video ID	Red

The channel value must be converted using an AND operation with the value 0x0F and the addition of the number

Table IV  
BYTE OFFSETS VIDEO FRAGMENT DESCRIPTOR

Offset	Description	Farbe
0x00 - 0x00	Status	Yellow
0x01 - 0x01	Channel	Orange
0x02 - 0x03	Internal ID of video fragments	Green
0x04 - 0x07	Start of record	Cyan
0x08 - 0x0B	End of record	Blue
0x0C - 0x0F	Next descriptor ID	Magenta
0x14 - 0x17	Previous descriptor ID	Pink
0x18 - 0x1B	Video ID	Red

1. For example, in Figure 5, the value 0x23 is converted to the value 4 via the calculation  $(0x23 \& 0xF) + 1$ .

The descriptor table makes it clear that the stored video recordings can be fragmented within the file system. Like FAT, it uses a cluster chain, which references the allocation status of a cluster as well as the clusters in the chain. Each partition has only one descriptor table; a copy could not be found. Therefore, a bitmap for determining the allocation status, as used in NTFS, for example, is not necessary..

#### B. File content category

The file content category includes all information required to determine the data. This includes, for example, the sizes of the allocation units and the so-called logical file system address.

The boot sector of a DHFS 4.1 partition defines the size of the bytes per sector and also the number of sectors per cluster. A cluster is therefore the allocation unit required for storing and determining the size of a video file.

The cluster numbering is identical to the numbering in the descriptor table. The first descriptor corresponds to the first cluster, the second descriptor to the second cluster, and so on.

Since the descriptor table is a doubly linked list of main descriptors and video fragment descriptors, the size of the data area can be determined. This is calculated by multiplying the number of existing descriptors by the number of sectors/clusters.

Clusters are numbered from logical file system address 0 to  $n - 1$ . Conversion to the logical block address (LBA) is performed by summing the offsets to the partition and data area, added to the product of the cluster size and the descriptor or cluster number. The formula is shown below:

$$LBA = Startofthepartition + Startofvideodataarea + (Clustersize * Descriptornumber) \quad (1)$$

Thus, the exact LBA can be determined for each cluster, which is essential for restoring fragmentation.

To determine which fragments are assigned to a main descriptor, all fragments from the main descriptor onward must be traversed like a linked list. The chain is fully determined once the number of fragments in the video is reached or the last fragment has no successor (End-Of-File, EOF).

The last fragment also has the special characteristic of varying size. The main descriptor specifies the size of the last fragment, so, for example, the last fragment might only reserve 3241 sectors instead of 4096. This condition can create a so-called slack space, which can be further analyzed.

The clusters, and therefore the LBA, can now be determined based on the descriptor number in order to extract a file from the file system..

It is not known exactly how the file system decides which descriptor is used next, and it seems to follow the order in which an actively written cluster is completely full. Because multiple channels are recorded simultaneously, the clusters of a video may not be used contiguously, resulting in fragmentation.

The technique for deleting or releasing descriptors is also not clearly defined. In most video surveillance systems, recordings are successively overwritten after a certain period, often 7 or 14 days. It is unknown whether deleting individual video recordings resets the descriptors to the free state (0xFE).

#### C. Metadata category

Metadata describes the data that resides on a file system. In this sense, it is „data around data“ and includes, for example, timestamps or addresses to the data areas that a file allocates.

In DHFS 4.1, only a small number of metadata exist: the recording date of a main video descriptor, usually within a one-hour period, and the timestamps of the associated video fragment descriptors, which can be narrowed down to minutes or seconds.

A direct logical data system address does not exist, however, as described in formula 1, the address can be determined by conversion using the descriptor ID.

1) *Timestamps*: The timestamps in DHFS 4.1 are each 32 bits in size and quite similar to UNIX timestamps, which are also 32 bits in size. UNIX timestamps are calculated in seconds starting from January 1, 1970, so every second from that point onward is counted. Therefore, one minute corresponds to 60 seconds, one hour to 3600 seconds, and so on.

For example, the date 11.08.2025 11:54:27 (GMT +2) corresponds to the 32-bit value 1754906067. Time zones are specified with Greenwich Mean Time (GMT) +X, so, for example, Central European Summer Time (CEST) is specified

as GMT +2.

The DHFS 4.1 timestamp calculates differently, one hour is equivalent to the value 4096. Therefore, to determine the exact time, i.e. year, month, day, hour, minute and second, the bit mask from table V must be applied to the 32-bit value.

Table V  
TIMESTAMP BIT MASK

Bits	Length	Description
31–26	6	Year
25–22	4	Month
21–17	5	Day
16–12	5	Hour
11–6	6	Minute
5–0	6	Second

However, the year only contains the last two digits of the full year, so the value 2000 must be added.

For example, the timestamp 0x76A31851<sub>LE</sub>, which corresponds to the decimal value 1360569206, is converted in table VI.

Table VI  
CALCULATION OF 1360569206

Position	Value
Year	20
Month	4
Day	12
Hour	10
Minute	13
Second	54

Adding the year 2000 to the date results in the readable date 12.04.2020 10:13:54. The time zone is assumed to be GMT +0, so the time zone offset must be added manually.

Aside from the timestamps, no further metadata is known. There are no access, creation, or modification timestamps like those found in NTFS or FAT file systems. It is also unknown why most video sequences are recorded within a one-hour period, unless the recordings are interrupted.

#### D. File name category

In DHFS 4.1, there are no direct filenames known from other file systems. Paths and folder structures were also not recognized. Instead, the files on a partition are arranged in a kind of flat list, so that all video files can be found in the (virtual) root path of a partition.

It is assumed that the filenames are a composition of various elements, including the partition number, the recording channel, the start and end dates of the video recording, and

the main video descriptor number. This scheme was also chosen for programming the X-Ways Forensics X-Tension, allowing the video files to be uniquely identified.

Only the program developed by DAHUA displays the supposed path to the video file. This path consists partly of the elements mentioned above, but also contains unknown numbers that cannot be assigned, see figure. 7.

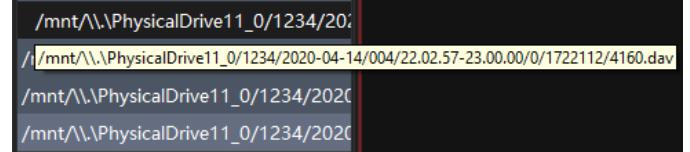


Figure 7. SmartPlayer file name

#### E. Application data category

Application data of a file system includes any data that is not relevant to the file system for obtaining data, but may be relevant for obtaining information about a file.

Typical application data includes file system journals, which, for example, record the state of a transaction (creation, writing) and can reverse it in case of inconsistencies.

DHFS 4.1 does not have a file system journal, but it does contain a log file that describes partition changes. Furthermore, the video files, which are stored in the proprietary DAV format, contain important information about the file structure, so these elements will be briefly discussed.

1) Log files: Each DHFS 4.1 partition contains an area for a log file. This can be read from the boot sector and, in all examined disk images, was sector 43520 from the beginning of the partition.

This sector contains a data structure that describes the log area of the file system, see figure 8. In this context it is called the log file header here.

00022282240	1F FD 05 00	00 00 00 00	E9 09 00 00	00 00 00 00	ÿ	€
00022282256	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282272	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282288	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282304	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282320	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282336	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282352	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282368	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282384	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282400	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282416	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282432	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282448	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282464	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282480	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282496	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282512	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282528	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282544	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282560	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282576	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282592	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282608	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282624	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282640	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282656	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282672	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282688	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282704	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282720	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00022282736	00 00 00 00	00 00 00 00	00 00 00 00	AA 55 AA 55	*0*U	

Figure 8. Log file header

```

00022283264 5B 48 44 44 4C 4F 47 5D 5B 30 5D 5B 31 38 2D 36 [HDDLOG][0][18-6
00022283280 2D 36 20 31 36 3A 30 3A 33 34 5D 5B 4F 70 65 72 -6 16:0:34][Oper
00022283296 61 74 69 6F 6E 5D 5B 30 2D 30 5D 20 66 6F 72 6D ation][0-0] form
00022283312 61 74 0A 5B 4B 44 4C 4F 47 5D 5B 31 5D 5B 31 [HDDLOG][1][1
00022283328 38 2D 36 20 31 36 3A 31 3A 35 37 5D 5B 49 8-6- 16:1:57][I
00022283344 6E 66 6F 5D 5B 30 2D 30 5D 20 49 6E 69 74 20 64 nfo][0-0] Init d
00022283360 68 66 73 2C 20 63 75 72 72 65 6E 74 20 30 2D 30 hfs, current 0-
00022283376 2C 20 67 5F 64 69 73 6B 5F 6E 75 6D 20 31 2C 20 , g_disk_num 1,
00022283392 5B 30 39 30 39 32 32 50 42 34 32 30 31 51 53 4B [090222P84201QSX
00022283408 54 33 4C 47 42 5D 20 54 79 70 65 3A 20 30 2C 20 TSLGB] Type: 0,
00022283424 65 72 72 6F 72 5F 66 6C 61 67 20 30 2C 20 74 6F error_flag 0, to
00022283440 74 6F 6C 20 63 6C 75 73 74 65 72 20 32 39 38 30 tol cluster 2980
00022283456 37 2C 20 63 75 72 65 6E 74 5F 63 6C 75 73 74 7, current_clust
00022283472 65 72 20 30 0A 5B 48 44 44 4C 4F 47 5D 5B 32 5D er 0 [HDDLOG][2]
00022283488 5B 31 36 2D 36 2D 36 20 31 36 3A 31 3A 35 38 5D [18-6- 16:1:58]
00022283504 5B 4F 70 65 72 61 74 69 6F 6E 5D 5B 30 2D 30 5D [Operation][0-0]
00022283520 20 53 65 74 44 72 69 76 65 72 54 79 70 65 20 35 SetDriverType 5
00022283536 0A 5B 48 44 44 4C 4F 47 5D 5B 33 5D 5B 31 38 2D [HDDLOG][3][18-
00022283552 36 2D 37 20 34 3A 31 30 3A 34 37 5D 5B 49 6E 66 6-7 4:10:47][Inf
00022283568 6F 5D 5B 30 2D 30 5D 20 43 60 61 6E 67 65 50 61 o][0-0] ChangePa
00022283584 72 74 20 66 72 6F 6D 20 5B 30 2D 30 5D 20 74 6F rt from [0-0] to
00022283600 20 5B 30 2D 31 5D 20 72 65 73 65 61 6E 20 30 0A [0-1] resean 0
00022283616 5B 48 44 4C 4F 47 5D 5B 34 5D 5B 31 38 2D 36 [HDDLOG][4][18-6
00022283632 2D 39 20 30 3A 31 35 3A 33 5D 5B 49 6E 66 6F 5D -9 0:15:3][Info]
00022283648 5B 30 2D 30 5D 20 43 68 61 6E 67 65 50 61 72 74 [0-0] ChangePart
00022283664 20 66 72 6F 6D 20 5B 30 2D 33 5D 20 74 6F 20 5B from [0-3] to [
00022283680 3D 20 3D 20 72 65 73 65 61 6E 20 30 0A 5B 48 0-0] resean 0 [H
00022283696 44 44 4C 4F 47 5D 5B 35 5D 5B 31 38 2D 36 2D 39 DDLOG][5][18-6-9
00022283712 20 31 33 3A 31 35 34 5D 5B 49 6E 66 6F 5D 5B 13:1:54][Info][
00022283728 30 2D 30 5D 20 43 68 61 6E 67 65 50 61 72 74 20 0-0] ChangePart
00022283744 66 72 6F 6D 20 5B 30 2D 30 5D 20 74 6F 20 5B 30 from [0-0] to [0
00022283760 2D 31 5D 20 72 65 73 65 61 6E 20 30 0A 5B 48 44 -1] resean 0 [HD

```

Figure 9. Log file data

The data structure contains only two known and essential values: the length of the log file in bytes and the number of log entries, see table VII. At the end of the sector, the magic number 0x55AA55AA can be found again, marking the end of the header.

A detailed log file analysis is outside the scope of this work, but it is assumed that the log files contain, among other things, information about the rotation of the partitions or previously existing recording data.

2) DHII data structure: Following the cluster chain of a main video descriptor from the first video fragment, using the calculation described in the section IV-B, one does not initially land directly on video data but on a sector containing another, descriptive data structure. This data structure is called a DHII data structure because the first 4 bytes of this structure contain the hexadecimal value 0x44484949, which corresponds to the ASCII representation „DHII“ (see figure). 10.

Table VII  
BYTE OFFSETS LOG FILE HEADER

Offset	Description	Color
0x00 - 0x03	Byte length (32 Byte)	Yellow
0x08 - 0x0B	Number of log file entries	Red
0x1FC - 0x1FF	Magic number	Green

Following the data structure is a 1:1 copy of the sector. After that, the log file data can be viewed. Unlike the video files, these are not fragmented and can be read in their entirety. The file content is plain text, see figure 9.

09155379200	44 48 49 49	00 00 06 00	01 00 00 00	01 00 00 00	DHII
09155379216	40 00 00	C8 BE 03	00 00 00 00	00 00 00 00	Ø
09155379232	00 00 00	00 00 00	00 00 00 00	00 00 00 00	È
09155379248	00 00 00	00 00 00	00 00 00 00	00 00 00 00	Ó
09155379264	80 08 06 00	54 08 00 00	55 E2 1D 51	00 00 00 00	Vb Q
09155379280	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379296	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379312	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379328	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379344	00 00 00	D3 1B 06 00	8F OB 00 00	56 62 1D 51	Ø
09155379360	00 00 00	00 00 00	00 00 00 00	00 00 00 00	ž
09155379376	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379392	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379408	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379424	00 00 00	00 00 00	00 00 00 00	16 30 06 00	9E OB 00 00
09155379440	57 62 1D 51	00 00 00	00 00 00 00	00 00 00 00	WB Q
09155379456	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379472	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379488	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379504	00 00 00	00 00 00	00 00 00 00	00 E 45 06 00	
09155379520	81 08 00	58 62 1D 51	00 00 00 00	00 00 00 00	Xb Q
09155379536	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379552	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379568	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379584	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379600	D8 58 06	00 96 OB 00 00	59 62 1D 51	00 00 00 00	
09155379616	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379632	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379648	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379664	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379680	00 00 00	00 00 00	00 00 00 00	00 00 00 00	
09155379696	00 00 00	00 00 00	00 00 00 00	00 00 00 00	

Figure 10. DHII data structure

This data structure consists of two parts: a header (64 bytes) and a list with entries of 84 bytes each. The number of entries can be found in the header starting at byte 0x14; see table VIII.

Table VIII  
BYTE OFFSETS DHII HEADER

Offset	Description	Color
0x00 - 0x03	Signature	Red
0x18 - 0x1B	Number of list entries	Yellow

A list entry in this data structure supposedly describes the exact position of a video frame within the entire video file. This requires three values: the offset from the beginning of the list structure, the length of the video frame, and the timestamp of the video frame. These can be read from the offsets in table IX. The remaining 72 bytes are not needed but separate the current entry from the next entries.

Table IX  
BYTE OFFSETS DHII LIST ENTRY

Offset	Description	Color
0x00 - 0x03	Offset to video frame	Cyan
0x04 - 0x07	Lenght of video frame	Green
0x08 - 0x0B	Start of record	Magenta
0x0C - 0x53	Unknown	Blue

It should be noted that the offsets are only valid for a defragmented video. The video file must therefore first be reassembled before the offsets can be applied to the video frames.

Tests showed that the DHII data structure is not required for video playback and, in fact, hinders it. It is sufficient to assemble the video file starting from the first video frame. The video frames are explained in the following section.

3) *Video frames:* Following the offsets in the list entry of the DHII data structure allows you to locate the individual video frames of a DAV video file. Figure 11 shows how the beginning of a frame is represented in the file system.

12588810240	44 48 41 56 FD 00 02 00	86 A2 04 00 BE OB 00 00	DHAvy	44 48 41 56
12588810256	00 70 1D 51	E5 11 14 FF	P	70 1D 51
12588810272	88 DF 66 E6 3B	00 00 00 94 02 00 00 00 00 00 01	Bfse;	"
12588810288	67 64 00 28 AC	E8 0E 81 26 C0 44 00 00 03 00 64	gd (→ è à d	d
12588810304	01 00 13 88 10 00	00 01 68 EE 3C B0 00 00 00	hi<	
12588810320	01 65 88 81 C0 0C	CC 6F C4 E0 52 6F F1 40 2A 86	e' À ioÀRøñø+	t
12588810336	21 37 EF F7 9D 3F	74 F5 38 8F 0D 34 08 E3 E5 3C	!71+ tç8 4 àk<	
12588810352	10 89 EB 00 00	03 00 00 03 00 00 03 00 00 03 00	hè	
12588810368	4A F4 D1 F9 39 C9 19 E3	45 A7 00 00 03 00 08 A3	JÖñùsè àës	£
12588810384	0C A7 3E CF BA 9A	20 78 B3 B7 66 25 37 0A 65 E8	S>i~ x' f%7 èè	
12588810400	03 B3 55 FA AE 61	12 7C 8A 4F EE AE 88 40 DB 38	'Uqøa  Søie~øøø	
12588810416	A5 OF 8C 78 F8 C8 78 A7	C1 37 5D AE 4C C4 B9 EF	# GxøExøSA7øLå.i	
12588810432	17 CC 5E 97 CD 5A	B3 A9 BE 67 B2 A7 B0 B8 67 22	i~Iz'ëgøs°.g"	
12588810448	44 FF 9A 25 15 01	8C 5D 9E 23 2C 53 D5 91 07 C9	Jyøs C]zø,øø' è	
12588810464	9E 8F 77 DA	C7 64 86 76 7F B6 22 98 B0 C9 75 6F	ž wÜcdtv q"~Éuo	
12588810480	DB 20 B5 40 F7	35 4F 0E 1E 67 34 B9 E3 C7 75 7C	Ø uø=5o gø~äçùl	
12588810496	0D 98 66 7A 66 C7 5F	F4 A0 90 9D 3D E2 C2 FE 45	"fzfç_ô =äPé	
12588810512	9E 16 11 76 93 BB	BB 0B 43 E5 49 EC 83 2B 95 AE	z v'ø~Cålf++ø	
12588810528	25 6E FO B7 17 9B	6A 4C F4 43 00 62 6F 0A 9E FA	ønø. ,jLøc bo žú	
12588810544	14 BB 41 76 4E	D8 66 33 63 68 C5 CE E5	»Avñømøø 03chñåä	
12588810560	C9 5A 4F 16 17 1C	59 99 C7 69 7E 50 77 69 B3 92	ÉZø Yøçí~Ewø'	
12588810576	8A 70 F9 5B 7B	6D D4 40 39 C3 8D D6 AB 27 1A A9	Spù{imøøøø C'ø	
12588810592	6D 71 36 01 47	EE AD 78 94 68 E3 6F AD 1B 16 5A	mq6 Gi-x"ñø~-Z	
12588810608	3C 3B 2C 95 F5	B9 E7 EB B3 27 7A C6 4F 55	<,ø~egø~ø~zZOU	
12588810624	5C A7 D5 C4 33 42	7B 27 6C DA BC 51 85 47 2C	\øöñB((1'ñQ,G,	
12588810640	14 BB 89 89 C7	B1 3F C6 1F 65 DD 6A 2D 82 B5 CE	_ñññçøzø eYj~,øù	
12588810656	87 0A 4E 17 EE	38 78 63 4F 34 2B E4 77 4A D9 DC	+ N iøxcO4+øwøøù	
12588810672	88 F7 C8 DD 65 68	E1 7A D3 CE F6 A8 2B 62 03 B3	-ÉYøhæzøfø+j,	
12588810688	E5 7D 8B C1 C1	A0 97 5D 04 09 E3 80 8D 82 B9 9D	À<ñ~ -] àe' ,	
12588810704	C4 EE BA 71 00 0A	0D D3 1D EA 95 04 EF 93 8F 15	Àiøq Õ è• i"	
12588810720	12 B3 FF B2 99 AB	B3 EA DC CC B2 91 D7 B3 43 9D	'yømøøøüiø~ø~c	
12588810736	FC 80 90 79 5F	EE F4 33 AC BA A4 56 77 84 27 F6	øø y_iø3~ø~øVwøø'ø	

Figure 11. Start of a DHAV video file / video frame

Little is known about the structure of a DHAV video frame, and the values determined here are based on assumptions. Table X lists some values that correspond to the values from the DHII data structure and the descriptor table.

Table X  
BYTE OFFSETS DHAV VIDEO FRAME

Offset	Description	Color
0x00 - 0x03	Signature	Yellow
0x06 - 0x07	Channel	Green
0x0C - 0x0F	Size of video frame	Cyan
0x10 - 0x13	Start or record	Red

The video frame's payload, presumably encoded in H.264, lies between the signatures at the beginning (0x44484156, DHAV) and at the end (0x64686176, dhav) (see figure 12). Further analysis of the payload or video data is outside the scope of this work and will not be discussed further.

12588813232	28 AA 01 C0 04 BC	64 68 61 76 BE OB 00 00 44 48	(* à ~dhavø	øh
12588813248	41 56 FC 00 02 00	87 A2 04 00 A7 00 00 00 70	Avø ~ø S	p
12588813264	1D 51 06 12 0C	F7 80 00 74 48 81 1A 08 19 94 02	Q ~ø tñ	"
12588813280	01 00 00 00 00 01	9A 24 43 33 1B FD 10 00 0F	øçç3 yñ	
12588813296	D9 C3 F5 F6 9E	22 53 BD 1A D9 06 DB 64 C2 OB A5	Üññz"ñø Ü ðññ	

Figure 12. End of a DHAV video file / video frame with subsequent DHAV signature

## V. SLACK SPACE AND CARVING OF VIDEO FILES

Now that the structure of DHFS 4.1 has been discussed, various forensic aspects can be addressed. The goal of any forensic investigation is the complete search for incriminating and exculpatory evidence, in this case, video data. Therefore, not only existing data, but also seemingly deleted data, come into focus for an investigator.

Analysis of the file system reveals two areas where an investigator can find further, possibly deleted, video files: in the slackspace of the last fragment of the video fragment list or by carving the clusters of free descriptor IDs. The terms slackspace and carving are briefly explained below.

Slack space is a data area that arises during the allocation and use of data units. For example, if two clusters of 1024 bytes each (a total of 2048 bytes) are reserved for a file, but only 1536 bytes are used, 512 bytes remain available. The situation becomes more interesting when the operating system does not overwrite the memory during allocation, thus preserving the contents of the previously existing file..

Carving refers to the process of program-based file recovery. [2] Many files, including DHAV video frames, have a fixed data structure with a header and a footer that mark the beginning and end of a file or data structure. During carving, a program, called a carver, searches for these signatures and applies various algorithms and recovery methods to restore the files without the aid of metadata. Fragmented files pose a challenge because there is no mapping between the file fragments and clusters, meaning that file content not belonging to the file being sought can be read between the header and the footer.

The following discusses the methods that can be used to find video files in the above-mentioned areas using DHFS 4.1.

#### A. Slack space of the last video fragment

Since the size of the last fragment in the video fragment list is known via the main descriptors, this area can be easily extracted and examined. Preferably, a carver can be used here to delimit the header and footer of a DHAV video frame.

#### B. Carving of free cluster

A simpler approach to carving is to search the clusters or slack space for the first occurrence of a header and then stop the search for that cluster. The information, including the timestamp of the video frame of all first occurrences, is analyzed and chronologically summarized, creating a new cluster chain. The underlying idea is to assume that a (former) video file has completely filled a cluster, since recordings typically last one hour.

The disadvantage of this method is that other DHAV headers are simply overlooked and grouped in the wrong date range, and/or the video file becomes unplayable or difficult to play due to jumps in the time.

A more efficient method is carving from individual video frames. A carver could locate all valid video frames and combine them, so that parts of a video can be reconstructed from the fragments based on the extracted timestamps and channels.

The aforementioned cluster fragmentation becomes problematic because various factors need to be considered. With a presumably high degree of fragmentation, a channel in a configuration of 512 bytes per sector and 4096 sectors per cluster describes a maximum of  $4096 * 512 \text{ bytes} = 2 \text{ MB}$ .

If a video frame has a size or offset within the cluster that extends beyond the cluster boundary, reconstructing that video frame would only be possible to a limited extent. The footer, which might reside in a different cluster, contains only a size reference that must be compared to a potential header.

Several assumptions can be made to find fragmented video frames:

- 1) The defined size in the footer must match the size in the header. If they don't match, the footer doesn't belong to the video, and vice versa.
- 2) The bytes truncated by fragmentation from the header onwards correspond to the offset in another descriptor until the footer is found.

If, for example, fragmentation results in 34 bytes of the header being missing for the full length, and the end signature, which has the same size specification, is found starting from the beginning of another, nearby descriptor with an offset of 34 bytes, there is a very high probability that the header matches the footer and vice versa.

In general, the record date should be validated for each video frame to avoid accidental header signature matches. Furthermore, an incomplete header should be located in close proximity to an incomplete footer.

Since the content between the header and footer is unknown, the carver is heavily dependent on the header or the DHAV signature. If this is missing due to an override, entire video frames cannot be found and extracted.

## VI. DEVELOPMENT OF THE X-WAYS FORENSICS X-TENSION

Based on the findings of the file system analysis, an X-Tension can now be developed for the forensic program X-Ways Forensics.

For development purposes, X-Ways Forensics provides developers with interfaces (APIs) that allow the X-Tension to interact with the program and vice versa. The result is a DLL (Dynamic Link Library) file, which is dynamically included and executed by the program.

The programming language used can be either C or C++. The X-Tension developed here uses a mixture of C and C++

program code, and therefore constructs from both languages.

First, C structs are defined that correspond to the data structures of DHFS 4.1. For example, the boot sector of a DHFS 4.1 partition is represented as in Listing 1.

```
1 struct DHFS4_1_Bootsector {
2     uint32_t beginTime;
3     uint32_t endTime;
4     uint32_t sectorSize;
5     uint32_t clusterSize;
6     uint32_t descriptorTableOffset;
7     uint32_t descriptorTableItemCount;
8     uint32_t dataAreaOffset;
9     uint32_t logsOffset;
10 };
```

Listing 1. Boot sector C struct

Further data structures are present in the source code and can be viewed there.

For the X-Tension to interact with the program and vice versa, certain functions must be exported to the DLL. These special functions, prefixed with XT\_\*, are then called at runtime by X-Ways Forensics, thereby initializing the X-Tension.

Functions with the prefix XWF\_\* can be called from the DLL to interact with X-Ways Forensics. For example, the function XWF\_Read(...) can be used to read a specific number of bytes from a certain position from data area and process them later.

Subsequently, the developer of an X-Tension is responsible for creating a directory tree and reading the data from the source (disk or image). Here the X-Tension API provides important functions, too, such as creating files and fill them with data.

Special attention must be paid to the fragmentation of video files. A „standard“ X-Ways X-Tension is unable to process fragmented files. For this, the X-Tension must be extended with functions from the Disk I/O X-Tension API.

An X-tension in this mode is responsible for filling the data into the files displayed to the user in the directory browser. Additionally, every sector access must be processed to display the requested data to the user.

The X-Tension for the DHFS 4.1 file system was designed to cover both functions. It can be used to generate the directory tree and then, in disk I/O mode, to display the data in a defragmented state to ensure full functionality.

After the X-Tension is executed, the existing and carved video files, if any, are displayed. These can then be viewed or extracted for playback with external video processing programs.

*1) Structure and process of the X-Tension:* To parse the file system, the individual components analyzed in the file system analysis must be read, cached, and processed so that a file tree can be generated at the end of the processing, which is then displayed in X-Ways Forensics. The following describes the X-Tension process.

- 1) Read the partition table
- 2) Read the boot sector of a partition
- 3) Read the descriptor table of a partition
- 4) Carve the free descriptors
- 5) Carve the slack space used descriptors

Steps 2 to 5 are repeated for each existing partition in the file system.

As mentioned previously, C structs were defined to read and store the individual data structures. These structs are populated and read sequentially. First, the partition table is read, followed by the respective boot sectors of the partitions. A separate folder is created for each partition so that the video files for that partition can be stored there, as shown in the figure 13. The number of sub-objects is specified after the partition name.



Figure 13. Partition folders in X-Ways Forensics

After reading the boot sector, the descriptor table can be located and evaluated. Empty descriptors are temporarily stored, as these will later be used for starting the carving process.

If a main video descriptor is found, the cluster chain is traversed for all fragments. The result is a list of fragments or video fragment descriptors that can be assigned to a main video descriptor.

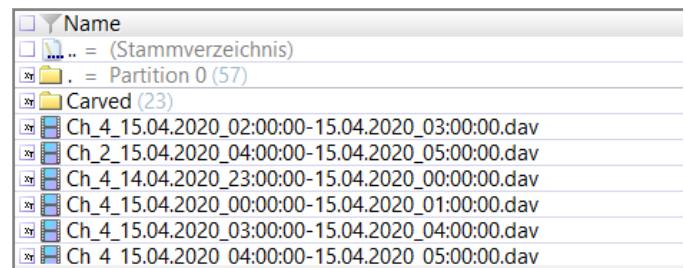


Figure 14. File names in X-Ways Forensics

The newly located file is immediately displayed in the directory browser of X-Ways Forensics. The file name is a combination of channel, start date, and end date to narrow down the time period of the files, see figure 14.

Additionally, metadata is added to the file, including the partition and the ID of the main video descriptor, see Figure 15. This metadata is necessary to fill the file with data, because the X-Tension is unloaded from memory after the file tree is built, but the metadata remains in the directory tree of X-Ways Forensics when active.

For log files or carved video files, the metadata includes „LogFile“ or „Carved“ to uniquely distinguish them. This is necessary because the data from carved and existing video files, as well as log files, are read in different ways.

Metadaten
0:11772
0:15758
0:5998
0:7926
0:13764
0:15756

Figure 15. Metadata in X-Ways Forensics

#### A. Carving of video files

Since the data area of the file system is fixed, carving is restricted to this area. Therefore, video data can only be found in clusters starting from the data offset. All sectors of free descriptors or clusters are searched for the signatures „DHAV“ and „dhav“. If, for example, the signature DHAV is found, it is first checked whether the following length and timestamp are valid. If the validation is successful, must be verified whether the length of the entire video frame extends beyond the boundary of the current cluster.

If the length is within the range of the current cluster, this video frame is added to a list. Otherwise, this video frame is marked as fragmented and stored in a separate list. If a dhav (i.e., a footer) is subsequently found, the list of fragmented video frames is searched backward until a matching header is found. If no header is found, the footer can be ignored as unassignable.

By searching for signatures and subsequently assigning them, entire video fragments can be generated from individual video frames. All video frames are assigned to a video fragment, which defines a specific hour, e.g., from 11:00

AM to 12:00 PM. The generated video fragments are then displayed in the directory browser of X-Ways Forensics in the folder „Carved“ of the respective partition.

#### B. Access to the video data

Since the files, except for the log files, are stored fragmented in DHFS 4.1, the X-Tension must be extended with the API functions of the Disk I/O. In this mode, the developer is responsible for filling the file buffer expected by X-Ways Forensics with data.

The buffer provided by X-Ways Forensics has a maximum size of 8 MB. The function for reading the data (XT\_FileIO) is called by X-Ways Forensics until no more data is available or the program returns an error.

The file is thus read sequentially. Fragmentation must be resolved, and the correct file offsets for reading must be determined. The program must always remember the current offset for each function call to resume reading at the correct position. Furthermore, it must be ensured that the offsets are reset for each new file. For this purpose, two global variables, ‘currentNItemID‘ and ‘moreFragmentsOffset‘, have been defined to ensure the aforementioned points are met.

## VII. CONCLUSION AND OUTLOOK

This paper describes the DHFS 4.1 file system from DAHUA. By analyzing the file system according to Carrier's structure, the file system could be decomposed into its individual components and analyzed. Based on this analysis, the X-Tension for X-Ways Forensics was developed.

It was demonstrated that the file system uses various structures from well-known file systems such as NTFS and FAT, and that the files within the file system are partially fragmented. Analyzing the descriptor table allows this fragmentation to be reversed, enabling the video files to be read and accessed.

The X-Ways Forensics X-Tension presented here serves as a basis for reading and extracting video files from DHFS 4.1. It demonstrates how a parser for the file system can be developed and used in principle. The carving method can potentially be extended and improved with information about the file structure of the DHAV files obtained in the future.

Based on the file system description, further new programs or extensions to existing programs can be developed. Some of these programs may be useful for law enforcement, as they could potentially facilitate the recovery of partially deleted video files to aid in the commission of crimes.

## REFERENCES

- [1] IBM. "What is digital forensics?" Accessed: Oct. 25, 2025. [Online]. Available: <https://www.ibm.com/think/topics/digital-forensics>.
- [2] B. Carrier, *File System Forensic Analysis*. Addison-Wesley, 2005.
- [3] J. Han, D. Jeong, and S. Lee, "Analysis of the hikvision dvr file system," vol. 157, Oct. 2015, pp. 189–199, ISBN: 978-3-319-25511-8. DOI: 10.1007/978-3-319-25512-5\_13.
- [4] N.-A. Le-Khac, R. Gomm, M. Scanlon, and T. Kechadi, "Analytical approach to the recovery of data from cctv file systems," Jul. 2016. DOI: 10.13140/RG.2.2.31446.65601.
- [5] W. Dongen, "Case study: Forensic analysis of a samsung digital video recorder," *Digital Investigation*, vol. 5, pp. 19–28, Sep. 2008. DOI: 10.1016/j.diin.2008.04.001.
- [6] E. Dragonas, C. Lambrinoudakis, and M. Kotsis, "Iot forensics: Investigating the mobile app of dahua technology," Jul. 2023, pp. 452–457. DOI: 10.1109/CSR57506.2023.10224982.
- [7] Magnet-Forensics. "Magnet witness," Accessed: Oct. 25, 2025. [Online]. Available: <https://www.magnetforensics.com/de/products/magnet-witness/>.
- [8] DiskInternals. "Diskinternals dvr recovery," Accessed: Oct. 25, 2025. [Online]. Available: <https://www.diskinternals.com/dvr-recovery/dhfs-4-1-file-system-dahua-nvr-cameras-data-recovery/>.
- [9] G. Batista. "Dhfs\_extractor," Accessed: Oct. 25, 2025. [Online]. Available: [https://github.com/gbatmobile/dhfs\\_extractor](https://github.com/gbatmobile/dhfs_extractor).
- [10] DmytroMoisiuk. "Dvr\_dahua," Accessed: Oct. 25, 2025. [Online]. Available: [https://github.com/DmytroMoisiuk/DVR\\_Dahua](https://github.com/DmytroMoisiuk/DVR_Dahua).
- [11] Dahua-Technology. "Smartplayer," Accessed: Oct. 25, 2025. [Online]. Available: [https://dahuawiki.com/Software/Dahua\\_Toolbox/SmartPlayer](https://dahuawiki.com/Software/Dahua_Toolbox/SmartPlayer).
- [12] Opengroup. "Seconds since the epoch," Accessed: Oct. 25, 2025. [Online]. Available: [https://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4\\_xbd\\_chap04.html#tag\\_21\\_04\\_16](https://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_xbd_chap04.html#tag_21_04_16).