
# Revisiting logical image formats for future digital forensics: A comprehensive analysis on L01 and AFF4-L

By:
Sorin Im, Hyunah Park, Jihun Joun, Sangjin Lee, Jungheum Park

From the proceedings of
The Digital Forensic Research Conference
**DFRWS APAC 2024**
Oct 22-24, 2024

Forensic Science International: Digital Investigation 50 (2024) 301811

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS APAC 2024 - Selected Papers from the 4th Annual Digital Forensics Research Conference APAC

# Revisiting logical image formats for future digital forensics: A comprehensive analysis on L01 and AFF4-L

Sorin Im [a], Hyunah Park [a], Jihun Joun [b], Sangjin Lee [a], Jungheum Park [a,*]

[a] School of Cybersecurity, Korea University, Seoul, South Korea
[b] School of Interdisciplinary Forensics, Arizona State University, AZ, 85306, USA

ARTICLE INFO

*Keywords:*
Digital forensics
Selective imaging
Logical imaging
Originality
Integrity
L01
AFF4-L

ABSTRACT

As the capacity of storage devices continues to increase significantly and cloud environments emerge, there is a need to perform logical imaging to selectively collect specific data relevant to a case. However, there is currently insufficient research addressing the appropriateness and usability of logical image file formats, which could potentially raise issues in terms of the originality and integrity of digital evidence. This study performs a comprehensive analysis of the internal structures and metadata of existing proprietary and open-source logical image file formats, with a particular focus on the L01 and AFF4-L. Furthermore, this study reveals several limitations of each file format and the supporting tools through practical experiments including metadata manipulation and stress tests. More specifically, the potential for loss of originality and metadata manipulation during and after logical imaging underscores the necessity for the development and standardization of more advanced logical image file formats to systematically manage different types of digital evidence from different sources. The findings of this research also demonstrate the necessity of collective efforts from the community for the continuous improvement of logical image file formats.

## 1. Introduction

A logical image is comprised of duplicates of one or more files, accompanied by metadata and file integrity details (Schatz, 2019). With the increasing capacity of storage devices, there is a growing necessity to selectively gather specific files. In environments such as the cloud, where physical imaging is not feasible, the significance of logical imaging is becoming increasingly apparent. In the United States, the Fourth Amendment and Federal Rule of Evidence 611 aim to limit the scope of seized items to "only what is directly relevant to the investigation", thereby minimizing privacy invasion and infringement of fundamental rights. Similarly, Europe's General Data Protection Regulation (GDPR) mandates that only necessary information be collected and processed during data handling. These regulatory frameworks emphasize selective information collection rather than broad evidence gathering, underscoring the importance of logical imaging. Tools such as EnCase, Tableau, and Falcon Neo have adapted to these requirements by incorporating logical imaging capabilities.

With the emergence of logical imaging, various specialized evidence file formats have been proposed for utilization. Commonly used logical image files in commercial tools include L01 and Lx01 from the OpenText EnCase product line and AD1 from the AccessData FTK product line. Additionally, an open-source form of logical image file proposed in academia is AFF4-L, which is also utilized in commercial products such as EnCase and Magnet AXIOM Cyber. In this context, we aim to contemplate the following three research questions (RQ).

- RQ1: Are the existing logical evidence formats adequately suited for the selective acquisition of potential digital evidence from various sources?
- RQ2: Are there any weaknesses or deficiencies from an evidence management perspective?
- RQ3: If any limitations exist, how can they be improved for supporting thorough logical imaging?

Therefore, this paper analyzes the L01 and AFF4-L image file formats and identifies the metadata items that can be stored in each format. Based on this analysis, integrity verification is conducted, and metadata manipulation and stress tests are performed to demonstrate the limitations of existing logical evidence formats from an evidence management

* Corresponding author.
*E-mail addresses:* lemonlaura@korea.ac.kr (S. Im), ha00116@korea.ac.kr (H. Park), jihun.joun@asu.edu (J. Joun), sangjin@korea.ac.kr (S. Lee), jungheumpark@korea.ac.kr (J. Park).

https://doi.org/10.1016/j.fsidi.2024.301811

Available online 18 October 2024

perspective. Furthermore, it emphasizes the need for collective efforts from the community towards developing and standardizing formats that enable thorough and secure management of potential digital evidence from various sources.

## 2. Background and related work

### 2.1. Forensic imaging and image formats

A forensic image is a collection of one or more files that encompass the contents and structure of a large storage device (Knight, 2011). Forensic image formats can be broadly categorized into RAW (dd: disk dump) and DEB (Digital Evidence Bag). The RAW format is a bit-by-bit copy of the target media. Hence, there is no additional metadata besides information about the original file, and various methods and file formats are required for different types of target media (Turner, 2005). In contrast, the DEB format, which serves as a universal container for digital evidence collection, is defined as a wrapper for gathering all types of digital evidence, storing both the original files and related metadata information (Turner, 2005).

The formats that implement the concept of DEB include EWF (Expert Witness Compression Format) and AFF (Advanced Forensic Format). EWF is the proprietary evidence compression format of OpenText EnCase, which includes formats such as E01, Ex01, L01, and Lx01. On the other hand, AFF is an open-source, scalable file format (Garfinkel et al., 2006), with variations like AFF4 and AFF4-L.

### 2.2. Selective collection and logical imaging

The traditional disk imaging process proceeds without error handling or logging capabilities from the beginning to the end of the target media collection (Turner, 2006). As storage devices have become larger in capacity, this approach requires significant time and storage space. Hence, recognizing that selective or partial acquisition may be more effective depending on the amount of information to be acquired, Logical Imaging(Selective Imaging) using the DEB format began to emerge (Turner, 2006). The concept of logical imaging was introduced by Turner in 2005 and implemented in 2006 (Garfinkel et al., 2006). Subsequently, the design and implementation of logical imaging using the AFF4-based container format have been proposed (Stüttgen et al., 2013; Faust et al., 2021).

Eco-Bag, a Merkel tree-based DEB container proposed in 2024, highlights the limitations of integrity verification in traditional logical image formats. It emphasizes the need for sophisticated and selective data collection techniques in modern digital investigations and presents an alternative to enhance the archival continuity and privacy protection of traditional logical imaging (Han et al., 2024). Furthermore, with the growth of the privacy field, research on selective collection and logical imaging for efficient privacy investigations continues (Halboob et al., 2015; Halboob,W., 2023).

However, most research has focused on proposing new logical imaging designs or introducing logical imaging frameworks to enhance privacy protection. There is a lack of in-depth discussion on how effectively existing logical image formats actually operate.

### 2.3. Logical image format

Representative logical image formats include EnCase's L01 and Lx01. EnCase began supporting logical imaging through the LEF (EnCase Logical Evidence File) format L01, based on EWF, from EnCase v5. From EnCase v7 onwards, it introduced the Lx01 format with LZ compression functionality. This LEF format was reverse-engineered by Joachim Metz in 2019 and implemented in the C language (Metz, 2019b).

Additionally, there are vendor-specific formats such as FTK Imager's AD1 and X-Ways' CTR, while well-known formats like TGZ, ZIP, VHD, etc., are also utilized by adding metadata information separately. However, pointing out the lack of a standardized format, academia proposed AFF4-L, an open-source format based on AFF4, in 2019 (Schatz, 2019).

AFF4 is a redesigned version of AFF, utilizing the Map-Stream approach and employing a container-based format, allowing multiple data streams to be stored within a single image file (Cohen et al., 2009). This makes it more efficient for storing and analyzing large-scale data. The Map-Stream method involves creating a virtual address space called Map, and navigating to specific addresses along this map reveals the existence of streams, thus virtualizing the storage space in such a manner. This supports the transition to non-linear block streams. Furthermore, AFF4 utilizes the standard format of RDF (Resource Description Framework) known as the turtle module to store metadata files and store a unique identity called the AFF4 URN (or ARN in the case of AFF4-L) for each object. AFF4-L, based on AFF4, also incorporates these features and additionally proposes AFF4-L by adding the deduplication functionality of Hash-based Imaging (Cohen and Schatz, 2010). It is implemented in the open-source pyAFF4 library (Cohen, 2019).

## 3. Reversing existing logical image file format internals

The L01 format provided by a well-known commercial tool 'EnCase' and the AFF4-L format supported by an open-source tool 'pyAFF4' have been selected as targets for this work's logical image analysis. Using EnCase and pyAFF4, sample logical image files are generated, and the structure of the image files is analyzed using the hex viewer program 'HxD'. Subsequently, a list of metadata provided by each image file is identified. In this process, EnCase is used for L01, while AFF4-L is analyzed using pyAFF4 as well as archiving tools. The datasets used for format analysis and metadata identification have been made publicly available online (Im, S., 2024). For ease of distinction, starting from here, the imaged files are referred to as Original File, and the outputs of the imaging process are referred to as Image File.

### 3.1. L01: a proprietary logical image format

An EWF file is a type of disk image that includes the contents and structure of an entire data storage device, a disk volume, or the physical memory(RAM) of a computer. The EWF file can take one of two formats: either as a bitstream (forensic image) or as a logical evidence file (LVF). L01 is a logical evidence file image format developed by Guidance Software (now part of the OpenText product line). It preserves the original file exactly as it exists on the media and documents attributes such as the assigned file name and extension, creation, modification, and last access dates and times, logical and physical sizes, MD5 hash values, permissions, start range, original path, and more (Family, 2015). In this paper, experiments and analyses were conducted using EnCase v22.3, the latest version as of May 2024.

#### 3.1.1. Internal structures of L01

The EWF format stores data across one or more segment files, with each segment file comprised of a 13-byte file header and one or more sections. Each section begins with a 76-byte section descriptor, containing information about the specific section. In EnCase, L01 files generated are composed of identical sections regardless of the filesystem or type of original file. A schematic representation of the overall structure of L01 has been posted on the online (Im, S., 2024), and more detailed information can be found in reference (Metz, 2019a).

The structure of L01 is mostly known as it is similar to EWF-E01. However, there are still several fields and category identifiers within the detailed data of each section whose meanings have not yet been determined. Among them, we have identified the contents of the map section - map entries array, as depicted in **Fig. 1**. The L01 map section consists of a section descriptor and a data area, which is further subdivided into a map string and a map entries array. The map string allows

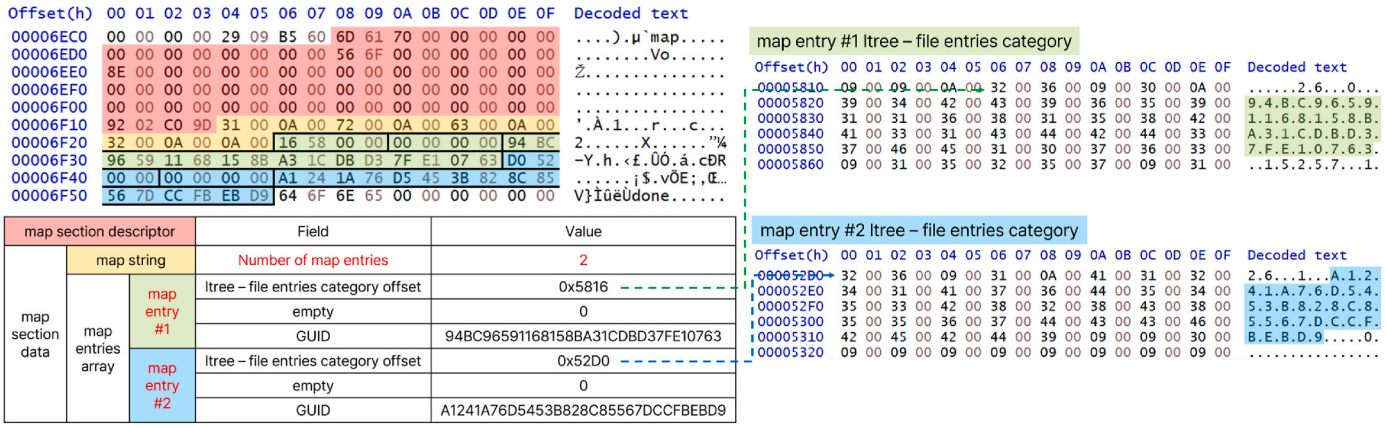



**Fig. 1.** Detailed analysis of the map section - map entries array.

us to determine the number of map entries, and there exist as many map entries in the map entries array as indicated by this count. Each map entry is 24 bytes long, with the first 4 bytes representing the offset within the ltree section - file entries category mapped to that map entry. The next 4 bytes are always an empty field, and the following 16 bytes represent the GUID of the file mapped to that map entry.

The category identifiers present in specific sections of L01 may vary depending on the EnCase version, thus requiring analysis for the latest version. In the L01 generated by EnCase v22.3, new identifiers that have not been previously identified were discovered, as shown in **Table 1**. Moreover, the meaning of the header2 section - sources category identifier has been ascertained. Upon opening an L01 file in EnCase, you can ascertain the 'Sources' information of the image file. The information is not within the internal structure of the L01 file, but rather exists in the Case directory - the directory of the corresponding evidence file GUID - in the 'EvidenceInfo.bin' file under the 'srl' category. This category follows the same structure as the sources category in the header2 section, allowing us to understand the meaning of the identifier in the header2 section - sources category.

#### 3.1.2. A list of metadata supported by L01

L01 provides metadata for both the image file and the original file. We imaged files existing in NTFS, FAT32, and exFAT file systems individually, including system files ($MFT), document files (.docx), compressed files (.zip), and folders, to verify their metadata.

**Table 2** presents a list of metadata for the image file, primarily found in the header2, header, volume, and data sections. The metadata remained consistent across all file systems and original file types. Interestingly, despite the experimental environment being Windows 11, the System Version was displayed as Windows 10.

Additionally, **Table 2** presents a list of metadata for the original file, primarily found in the ltree section, with slight variations depending on the file system. Variations were also observed based on the type of original file. When imaging system files and folders, the 'File Ext' field was not present. When imaging document files and folders within NTFS, the 'Logical Sequence Number' field was not present, while imaging system files and compressed files did not contain 'Object Identifiers'.

**Table 1**
Novel identifiers discovered in the EWF-L01 file structure.

| Section | Category | Identifier | | | |
|---|---|---|---|---|---|
| header2 | Source | do | ip | se | ma |
| | | loc | si | mfr | dt |
| | | mo | | | |
| ltree | Records | cd | md | wd | ad |
| | file entries | sha2 | spth | hshs | |

### 3.2. AFF4-L: an open-source logical image format

AFF4-L was introduced at DFRWS USA 2019 and Magnet Forensics, 2020. In 2020, Magnet Forensics began supporting AFF4-L by releasing AXIOM 4.5 and AXIOM Cyber 4.5 versions (Magnet Forensics, 2020). Additionally, EnCase has been supporting AFF4-L since the release of EnCase v22.3 in August 2022 (OpenText, 2022). Moreover, the previous version of Winpmem, 'winpmem_v3.3.rc3.exe' (Velocidex, 2019), can also be utilized for AFF4-L. However, since AFF4-L has been implemented as an open-source, extensible file format, each AFF4-L image file generated by different tools exhibited differences depending on the features of the respective tool. Therefore, the analysis of AFF4-L was conducted based on AFF4-L image files generated using pyAFF4, an open-source implementation for the AFF4-based logical imaging standard.

pyAFF4 was implemented with a focus on developing the logical imaging standard based on AFF4 (Cohen, 2019), and it validated its implementation by generating standard logical images and Hash-based logical images with duplicate removal functionality (Schatz, 2019). As of the first half of 2024, this open-source software operates on pyAFF4 version 1.1, with no updates since 2021. It supports functionalities such as reading and creating logical images, reading and creating deduplicated logical images, examining metadata of image files, extracting data, and generating encrypted AFF4 logical volumes (Cohen, 2019).

However, there have been instances where imaging did not proceed as expected due to errors occurring during the runtime of the open-source software. In pyAFF4, when the size of the original file exceeds 1024 KB during imaging, the file is compressed and stored in units called chunks. However, the code to define the length of the chunk was missing, making it impossible to image files larger than 1024 KB in size. Therefore, to facilitate smooth experimentation, we modified the code to define the length of the chunk during chunk creation. Moreover, duplicate removal imaging (Hash-based Imaging), faced difficulties when imaging a large number of files or files with large sizes, resulting in unsuccessful imaging. Therefore, the analysis and testing of AFF4-L in this paper focused on standard logical imaging.

#### 3.2.1. Internal structures of AFF4-L

Using the pyAFF4, we performed standard logical imaging of 'hello.txt' to generate 'test.aff4'. Upon examining the hex values, as shown in **Fig. 2 (A)**, we observed the ZIP file format signature. Subsequently, applying the file compression/decompression program 7-Zip revealed that the image file, as shown in **Fig. 2 (B)**, comprises the original file, 'container.description', 'information.turtle' and 'version.txt'.

For original files, the imaging process stores the path of the input original file exactly as entered, thus allowing the identical verification of the original file's tree structure within the file system if absolute paths

**Table 2**
A list of metadata supported by L01.

| File Type | Category | Metadata | | | |
|---|---|---|---|---|---|
| Image File | Image File (.L01) | Name | CRC Errors | Drive Type | Primary Path |
| | | Compression | System Version | Evidence Paths | Source Type |
| | | Sources | GUID | Case Number | Open Mode |
| | | Index File | Examiner Name | Evidence Number | File Integrity |
| | | Notes | Geometry | Application | Processing Status |
| | | Network Port | EnCase Version | Time Zone | Total Sectors |
| | | Error Granularity | Cache Status | Bytes Per Sector | Read Errors |
| | | Read File System | Parity Level | Missing Sectors | Parse Link File |
| | | Interface | – | – | – |
| | Evidence Paths | Name | – | – | – |
| | Sources | Name | Serial Number | Model | Total Bytes |
| | | Device GUID | Primary Device GUID | Drive Type | – |
| Original File | Original File | Name | Item Path | Evidence File | File Ext |
| | | True Path | File Identifier | Logical Size | Description |
| | | GUID | Category | [a]Entry Modified | [b]Short Name |
| | | Last Accessed | Initialized Size | Is Internal | File Created |
| | | Physical Size | Attributes | Last Written | File Extents |
| | | MD5 | Permissions | – | – |
| | Source | Name | Total Bytes | Drive Type | Serial Number |
| | | Device GUID | Model | Primary Device GUID | – |
| | [a]Attributes | Sequence ID | Logical Sequence Number | – | – |
| | [a]Object Identifiers | Own ID | Birth Volume ID | Birth Object ID | – |
| | [a]Permissions | Name | Property | ID | Permissions |

[a] Only available for NTFS.
[b] Only available for FAT32.

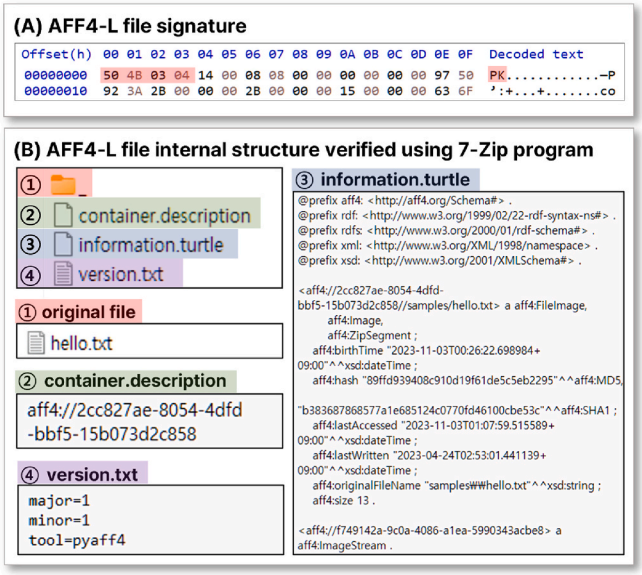



**Fig. 2.** AFF4-L file structure.

were provided during imaging (Schatz, 2019). In the case of Fig. 2 (B), by entering the path of the original file as a relative path, the original file could be located under the 'samples' folder. The three files excluding the original file are automatically generated during AFF4-L imaging. 'container.description' holds the container ARN information, thereby serving as the identifier of the AFF4-L file. 'information.turtle' stores the metadata of the original file in RDF format. The metadata information that can be verified here matches the information obtained using the '-m/–meta' option of the pyAFF4 open-source tool. 'version.txt' stores the version information of the pyAFF4 tool.

Fig. 3 depicts the format structure of the analyzed AFF4-L file ('test.aff4'), based on the structure of the analyzed ZIP file format and regarding the pyAFF4 open-source tool. Each file within the ZIP file format contains a 'Local File Header' and a 'Central Directory' At the bottom of the format lies the 'End of Central Directory Record'. When the file is executed, it operates by reading from the 'End of Central Directory Record' at the bottom to the position of the 'Central Directory'. From here, it reads the position of the 'Local File Header' to retrieve the file's data. The AFF4-L file operates similarly, with the difference being the inclusion of the AFF4 ARN at the bottom of the file and the 'Flag' values of the 'Local File Header' and 'Central Directory', which are both set to '0x0808'. Considering that the ZIP file format does not define the Flag value as '0x0808' and the Flag value for AFF4 files is '0x08', it can be inferred that '0x0808' represents the Flag value for AFF4-L. Furthermore, upon examining Fig. 3, it was observed that the data section following the 'Local File Header' is exposed as-is in HxD, without compression or encryption, except for the original file.

In the case of Fig. 3, referencing the 'test.aff4' file, there are four instances of both 'Local File Header' and 'Central Directory'. However, if there are numerous original files imaged or if the size of the original file is large enough to be split and stored across multiple files, additional instances of 'Local File Header' and 'Central Directory' are generated corresponding to the number of files.

### 3.2.2. A list of metadata supported by AFF4-L

The metadata of AFF4-L is stored in RDF format in 'information.turtle', which can be inspected using open-source tools or compression programs. The metadata information provided by both methods is identical, and Table 3 summarizes the metadata supplied by AFF4-L. The metadata is generated in two cases based on the original file's size. When imaging files smaller than 1024 KB, 8 metadata items are created, including ARN, path/name of the original file, size, MAC timestamp, and hash value. For files larger than 1024 KB, the original file is divided into chunks and undergoes split compression during imaging. Therefore, in addition to the existing metadata, metadata regarding chunk information and compression method are added, resulting in a total of 11 metadata items.

Fig. 4 depicts the metadata retrieved from the AFF4-L file 'information.turtle' after imaging files smaller and larger than 1024 KB. After imaging the 1 KB-sized 'hello.txt' file to 'test.aff4', metadata inspection reveals a total of 8 metadata items, as shown in the upper image of Fig. 4. Next, after imaging the 1025 KB-sized '1025 KB.txt' file to '1025 KB.aff4', metadata inspection displays 11 metadata items, including additional metadata regarding chunks and compression method, as depicted in the lower image of Fig. 4.

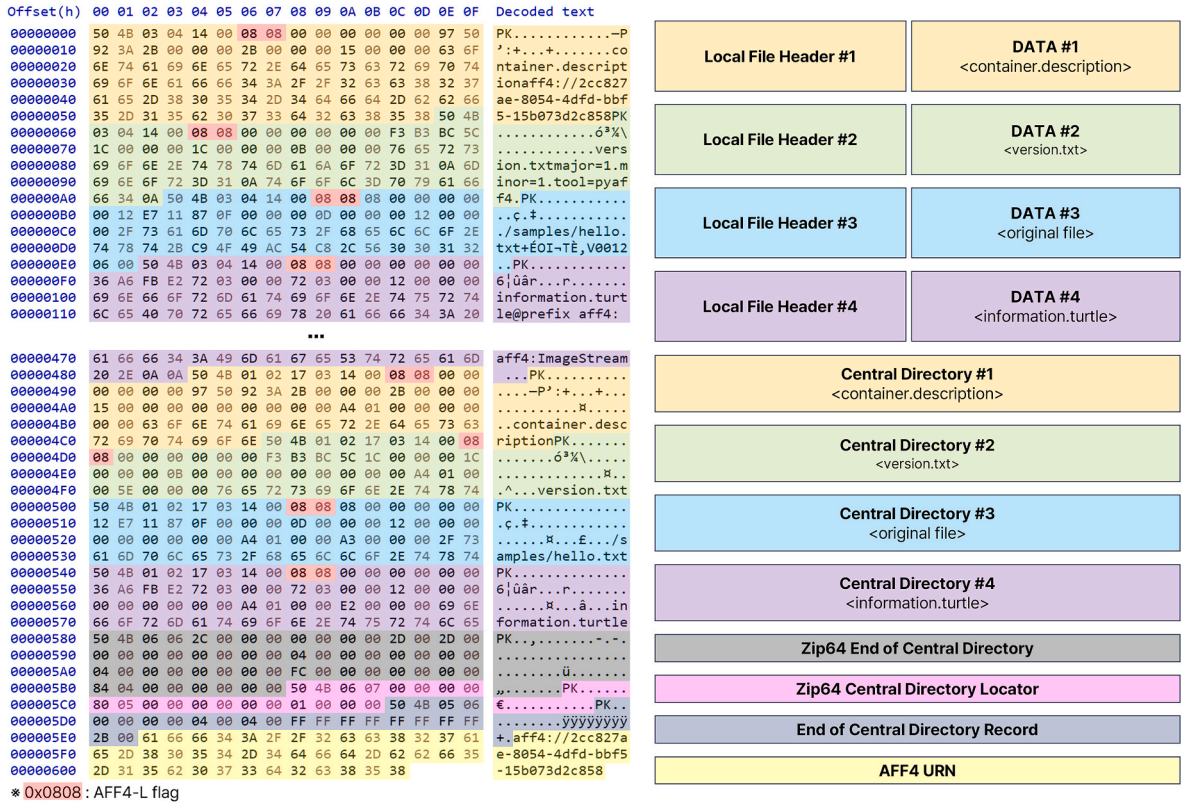



**Fig. 3.** AFF4-L file format.

**Table 3**
A list of metadata supported by AFF4-L.

| File Size | Metadata |
|---|---|
| 1,024 KB or less | Image Stream ARN<br>File Image ARN<br>Original File Path Name<br>Original File Size<br>Original File Birth Time<br>Original File Last Accessed Time<br>Original File Last Written Time<br>Original File Hash (SHA1, MD5) |
| Over 1,025 KB<br>(Additional generated metadata) | Chunk Size<br>Chunk In Segment<br>Compression Method |

Metadata verification was conducted for 58 files with various extensions including documents, images, audio, video, and system files, besides ‘txt’ files. However, no peculiarities were observed for any file, with all providing the same type and quantity of metadata information. Additionally, logical imaging was performed for FAT32 and exFAT files, in addition to NTFS, to examine filesystem-specific characteristics. Nevertheless, no filesystem information or peculiarities were identified.

## 4. Analyzing the limits of existing logical image formats

To analyze the limitations of the existing logical image formats L01 and AFF4-L, the metadata identified in Section 3 was examined to investigate any loss of originality. Additionally, the integrity verification information for each format was reviewed, and metadata manipulation tests were performed to assess any failure of integrity verification. Finally, stress tests were performed to evaluate and compare the performance of the tools supporting each format.

### 4.1. Loss of originality with limited metadata types

In Section 3, we imaged and analyzed the metadata of original files with various extensions and sizes across NTFS, FAT32, and exFAT file systems. For AFF4-L, the types of metadata stored were identical regardless of the file system, thus failing to provide the diverse metadata that different file systems typically contain. Furthermore, despite the variety of original file types, the metadata provided by AFF4-L remained consistent. For instance, the metadata for image files included only ARN information, thus making it impossible to ascertain details such as the file name, imaging date, or examiner information. Similarly, metadata for original files did not adequately capture characteristic metadata information for various types of files, such as file system details or system files. L01 provides comparatively more metadata information and file system-specific metadata than AFF4-L. However, due to its policy of interpreting and normalizing the original metadata before storing it, L01 does not capture all of the original metadata. Notably, despite the variety of original file types, the metadata provided by L01 is largely similar, and it does not adequately reflect file system types or timestamp metadata.

Timestamps, which serve as the foundational evidence for analyzing events in chronological order, are among the most fundamental yet crucial pieces of information in digital forensics analysis (Bang et al., 2011). Each file system contains various timestamp information, and both L01 and AFF4-L metadata include multiple timestamp details. However, there are limitations as not all timestamp information is included. For instance, NTFS uses FILETIME format, which can express time up to seven decimal places, while L01 uses the Unix Seconds format, limiting storage to seconds, and AFF4-L provides timestamps up to six decimal places. Table 4 outlines the timestamp information not provided by L01 and AFF4-L, highlighting the need for their inclusion in digital forensics analysis.

Both formats use the DEB format to interpret the original metadata and normalize it to the specified metadata before storing it. However,

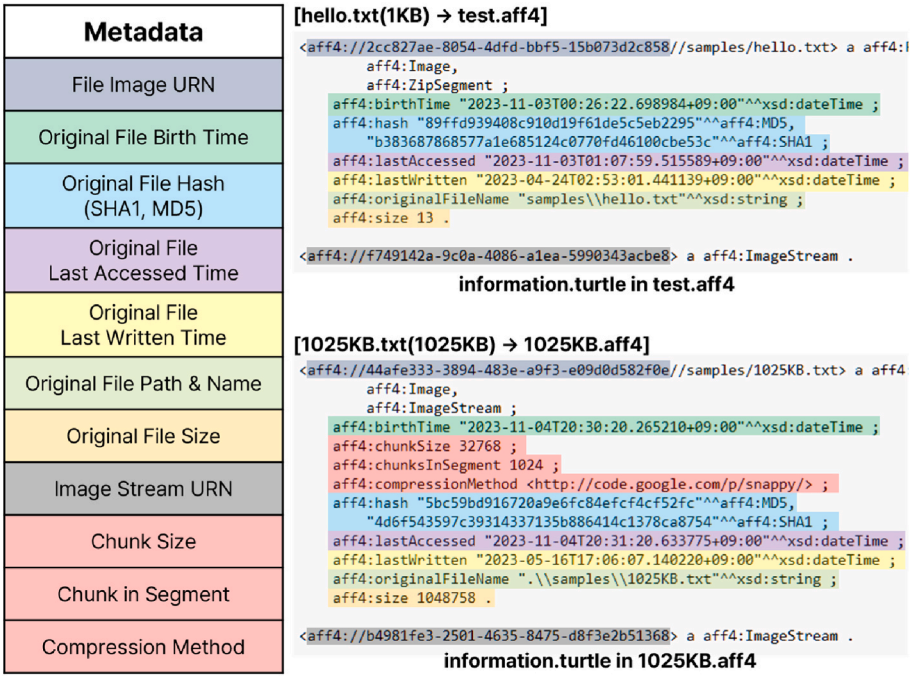



**Fig. 4.** AFF4-L metadata comparison.

**Table 4**
Timestamps to be added to logical images (L01, AFF4-L).

| Format | File System | Attribute | Data | Description |
|---|---|---|---|---|
| L01 &AFF4-L | NTFS | $FILE_NAME | Created Time | File creation time |
| | | | Modified Time | File modification time |
| | | | MFT Modified Time | MFT Item modification time |
| | | | Last Accessed Time | Last time a file was accessed |
| Only L01 | FAT32 | Directory Entry | Create Time Tenth | File creation time in 1/10 units |
| | exFAT | File Directory Entry | Create 10 ms | Generation time in 10 ms |
| | | | Last Mod 10 ms | Last modified time in 10 ms |
| Only AFF4-L | NTFS | $STANDARD_ INFORMATION | MFT Modified Time | MFT Item modification time |

since logical imaging does not collect the entire file system, it cannot collect raw data and must rely on the metadata information provided by the format. As a result, L01 and AFF4-L do not contain enough metadata information from various sources during the imaging process and lose their originality. To address these issues, improvements are needed to enable archiving and managing potential evidence from multiple sources, such as different file systems or cloud data, including storing raw data (e.g., NTFS's $MFT Entry) together so that all available metadata can be utilized later.

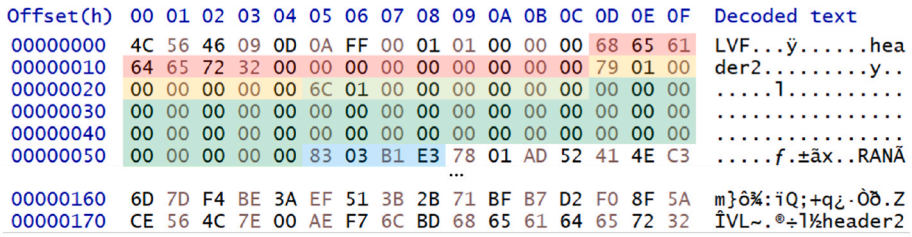
### 4.2. Failure of integrity verification against metadata manipulation

After analyzing each format, we found that both L01 and AFF4-L lack integrity verification information and security evaluation metrics. First, neither format provides a representative verification value at the format level to demonstrate integrity. This is in contrast to E01, the leading digital forensic image format, which provides integrity verification information via a hash value of the file upon completion of imaging. In addition, both formats are simple and unorganized in structure. Without encryption, individual metadata fields can be exposed through a simple analysis using a hex editor, and the simplicity of the format makes it relatively easy to calculate checksums and verify CRC values. Therefore, the integrity of each format was tested by manipulating the metadata of the L01 and AFF4-L formats and checking the ability of relevant tools to detect the manipulations. Two ways of manipulation are defined: *Simple Manipulation*, where only the metadata is altered, and *Elaborate Manipulation*, where both the metadata and CRC value are altered. All metadata used in the L01 and AFF4-L manipulation are publicly available online, and detailed manipulation results for each format are available online.

#### 4.2.1. Integrity verification test of L01

For the integrity verification of the L01, we focused on selecting high-priority metadata for digital forensic analysis and conducted metadata manipulation tests accordingly. We categorized manipulation into three types: changing the value of the target metadata to a different value of the same length, completely deleting the existing value of the metadata, and adding arbitrary values to consistently empty metadata entries.

Each section in L01 starts with a section descriptor structured as shown in Fig. 5. The checksum of the section descriptor is calculated using Adler-32 checksum for all data within the section descriptor, from Section Type to padding, thereby verifying the integrity of the section. Therefore, when performing actions such as deleting or adding metadata values resulting in changes in the length of the metadata, the next section offset, section size, and checksum values within the section




| Field Name | Offset | Size | Value | Description |
|---|---|---|---|---|
| Section type | 0 | 16 | | A string containing the section type definition |
| Next section offset | 16 | 8 | | Next section Offset |
| Section size | 24 | 8 | | Section size, including section descriptors |
| Padding | 32 | 40 | 0x00 | |
| Checksum | 72 | 4 | | Adler-32 of all data in section descriptors |

**Fig. 5.** Detailed analysis of the section descriptor.

descriptor of that section, as well as the next section offset and checksum values within the section descriptors of all sections following it, must be recalculated.

Certain sections contain additional integrity verification elements within their section data, and Table 5 lists them. When modifying metadata values within these sections, the integrity verification elements existing within the section data must also be recalculated. L01 metadata manipulation tests were conducted, distinguishing between Elaborate manipulation, which recalculates all elements affected by the manipulation actions mentioned earlier, and Simple manipulation, which does not involve such recalculations.

When creating an L01 file, you can set options for compression and Entry Hash. If the compression option is selected, the Sectors section data exists in a zlib-compressed state, whereas if compression is not selected, the Sectors section data remains uncompressed, identical to the original file. When the Entry Hash option is enabled, the data paired with the 'ha' identifier in the ltree section under the file entries category, is converted into the MD5 hash value of the original file. If the Entry Hash option is not enabled, the corresponding data is set to 0, indicating it is "not set" Accordingly, the sample file is structured as shown in Table 6.

Since the header2 and header section data are compressed with zlib, to manipulate the metadata of these sections, it is necessary to go through the process of decompression, metadata manipulation, and recompression. To derive accurate results from metadata manipulation tests, the compression options used during compressing the section data were identified, as shown in Table 7.

Even if the values of the target metadata are changed to different values of the same length, the length of the recompressed data may differ from the original. Therefore, when manipulating the metadata within the section, regardless of the action taken, the checksum values and next section offsets within the section descriptor of that section, as well as those of all subsequent sections, must be recalculated.

Based on the analysis conducted so far, attempting to manipulate the metadata of the header2 and header sections results in a verification failure message in EnCase, stating 'The integrity of the following sector groups could not be verified'. It suggests that there are unidentified integrity verification fields and values requiring further investigation. However, since EnCase did not detect any manipulation within the original file, which contains the actual evidence data, this study focused on this aspect.

The volume section data forms a unique structure in its entirety. Therefore, if the GUID of the volume section is completely manipulated with a value of the same length, it will be displayed as the changed value in EnCase. However, if the GUID is deleted, even with a elaborate manipulation, an 'Invalid block checksum' error occurs.

The ltree section data is divided into five categories, among which the identifiers–data pairs of the sources category and the file entries category were selected for the manipulation test. The sources category stores the metadata of the original file sources, while the file entries category stores the metadata of the original files.

If the Name of the original file source and the File Created value of the original file are fully manipulated to values of the same length,

**Table 5**
Integrity verification elements for each section.

| Section | Integrity Verification Element | Description |
|---|---|---|
| Volume | Checksum | Adler-32 of all data (from Media Type to Reserved)in the volume section |
| Table | table header - Checksum | Adler-32 of all data in the table section |
| | table footer - Checksum | Adler-32 of an offset array |
| | header - Checksum | Adler-32 of all data in the table section |
| | footer - Checksum | Adler-32 of an offset array |
| Ltree | ltree header - MD5 | MD5 hash of entire ltree data, big endian |
| | ltree header - Checksum | Adler-32 checksum of all data in ltree header (excluding unknown field) with checksum value of 00000000 |
| Data | Checksum | Adler-32 of all data in the data section (from Media Type to Reserved) |
| Done | Checksum | Adler-32 of all data in the done section (from Section Type to Padding) |

**Table 6**
L01 sample file for metadata manipulation test.

| Entry Hash | Compression | |
|---|---|---|
| | Disabled | Enabled |
| None | origin.L01 | originCom.L01 |
| MD5 | originMD5.L01 | – |

**Table 7**
Compression options for header2 and header sections.

| Option | | Value | Description |
|---|---|---|---|
| CMF | CM | 8 | deflate compression method with a window size up to 32K |
| | CINFO | 7 | 32K window size |
| FLG | FCHECK | 1 | check bits for CMF and FLG |
| | FDICT | 0 | no preset dictionary |
| | FLEVEL | 0 | compressor used fastest algorithm |

EnCase will display the modified values. Conversely, if these are fully deleted, the corresponding entries existing in the original EnCase data will also be deleted. When the GUID of the original file is fully manipulated to an arbitrary value of the same length, EnCase will display the changed value. However, if this GUID is fully deleted, the corresponding entry will not be deleted; instead, a new GUID value will be displayed.

In the case of MD5 and SHA1 values of the original file source, they are always stored as 0, indicating "not set". If these are fully manipulated to values of the same length, new MD5 and SHA1 entries will be added to EnCase, which were not present before, and will be displayed with the modified values. Similarly, 'Acquisition date and time' of the original file source and 'Deletion date and time' of the original file always exist as empty values. If arbitrary values are fully added for each, new Acquired and Deleted entries, which were not present before, will be added to EnCase, and displayed with the manipulated arbitrary values.

The option to set the Entry Hash when creating an L01 file was previously mentioned. Although EnCase offers only None and MD5 options, the ltree section - file entries category includes an identifier sha corresponding to SHA1 in addition to the MD5 identifier ha. When MD5 is selected, the data paired with the identifier ha represents the MD5 hash value of the original file. If this is altered to a value of equal length, it will be displayed as the changed value in EnCase. Choosing the None option results in the paired data with the identifier ha representing 0, indicating it is not set. Altering this to a value of equal length adds a previously nonexistent MD5 entry to EnCase and displays the changed value. Moreover, since EnCase lacks the option to calculate the SHA1 hash value of the original file, the data paired with the identifier sha always remains 0. Nevertheless, altering it to a value of equal length adds a SHA1 entry that was not present in the original EnCase and displays the changed value.

When attempting to open the ltree section after performing simple manipulation by altering each target metadata to an arbitrary value of the same length, EnCase does not respond. In cases of simple deletion and simple addition, EnCase displays warning messages such as "Invalid block checksum" and "File not found". However, the outcome is similar to that of perfect deletion and perfect addition.

The data and map sections, similar to the volume section, constitute a single unique structure for their entire section data and include GUIDs. However, even when completely altered or deleted with values of the same length, they retain the original values without displaying any changed values or errors.

### 4.2.2. *Integrity verification test of AFF4-L*

Metadata manipulation in AFF4-L was conducted by dividing the process into two methods: image file metadata manipulation and original file metadata manipulation. Files for manipulation tests were prepared by imaging 'origin.txt' as 'origin.aff4' and 'target.txt' as 'target.aff4'. The manipulation content and results are presented in Fig. 6.

In AFF4-L files, there are two types of ARNs: Image Stream ARN and File Image ARN. The Image Stream ARN is located in 'information.turtle', while the File Image ARN is located at the bottom of the file format under 'container.description', 'information.turtle'. Therefore, a simple manipulation, as depicted Fig. 6 (A), was performed by replacing the ARNs at each location in the 'origin.aff4' file with the ARNs of 'target.aff4'. Subsequently, when the 'origin.aff4' file was checked using the pyAFF4 open-source tool, the image file could be read without any issues, and the manipulated content could also be confirmed. However, when checked using a compression program, the file could not be verified due to CRC value errors. In the case of AFF4-L, each file has a Local File Header and Central Directory with the same Checksum value. This value is calculated using the CRC-32 Checksum for the data following the Local File Header, ensuring the integrity of each file. Therefore, we calculated the CRC values of 'container.description' and 'information.tutle' using the manipulated data and manipulated the CRCs of 'Local File Header' and 'Central directory', respectively, to perform elaborate manipulation, as depicted Fig. 6 (B). As a result, the file could be verified without any issues using a compression program, and the manipulated content could also be confirmed.

Next, for manipulation with the original file metadata, the original file name in the 'origin.aff4' file was manipulated with to 'target.txt',

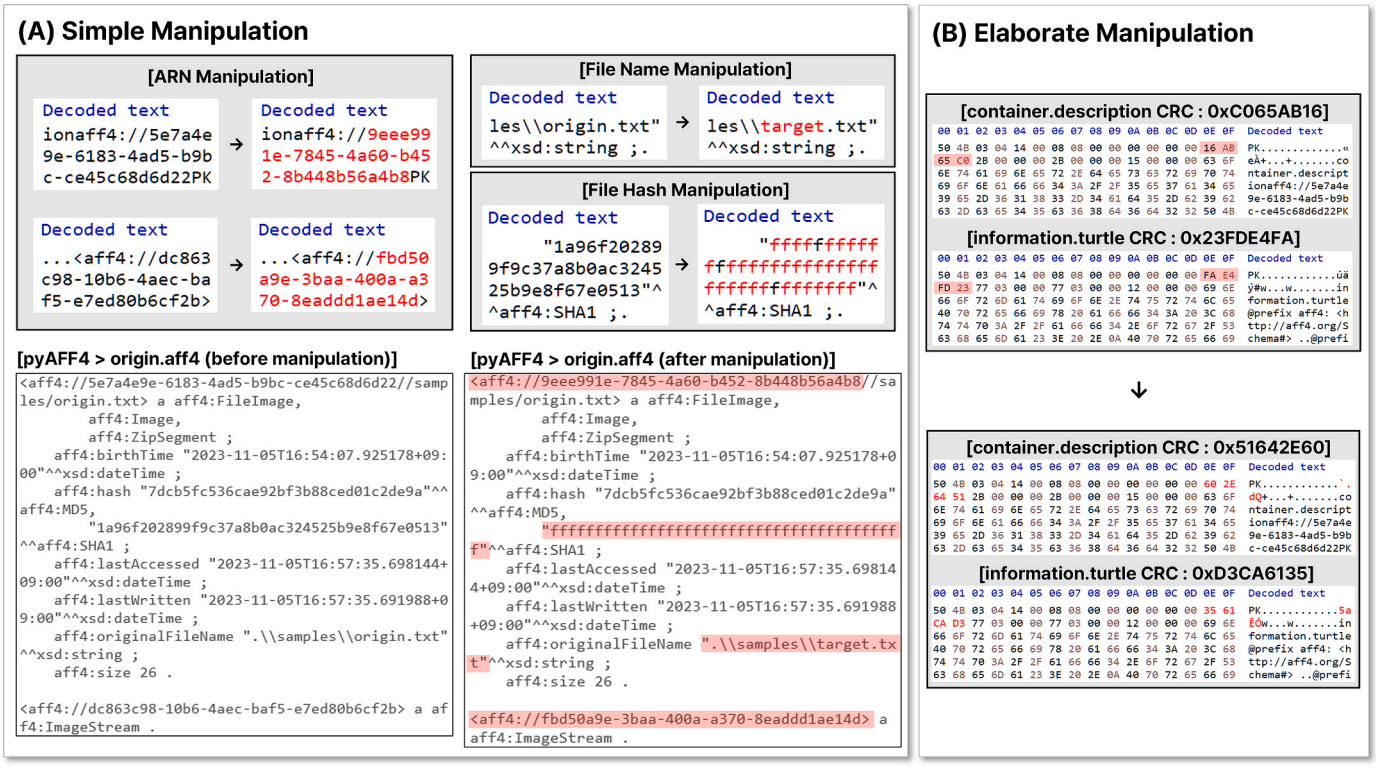



**Fig. 6.** Overview of AFF4-L metadata manipulation.

and the SHA1 hash of the original file was manipulated to all '0xff'. As a result, similar to the ARN manipulation, when simple manipulation was applied, it only worked in the pyAFF4 open-source tool, but when elaborate manipulation was performed, it worked without any issues even in compression programs.

In addition, for all existing metadata such as file size and MAC creation time, manipulation tests were conducted. The results showed that both simple manipulation and elaborate manipulation were possible for all metadata information that can be verified in HxD in the case of AFF4-L.

### 4.3. Lack of performance testing results including stress testing

Logical image formats are widely used, but there is no systematic performance testing efforts for them. The stress tests performed in this study fall into two categories: imaging a large number of files simultaneously and imaging large files. The same set of sample files was used to generate both L01 and AFF4-L image files. Comparative analysis was then performed on the image files to identify any unexpected results. The sample files used for this testing is publicly available online (Im, S., 2024). For AFF4-L, tests were conducted for both the default logical imaging method and the hash-based logical imaging method for duplicate removal. The results are summarized in **Table 8**.

To ensure no issues arose when imaging a large number of files, imaging was conducted for all folders and files within the 'Carpe' directory. This directory contained a total of 51,417 files with a combined size of 18.9 GB. Imaging with L01 took approximately 12 min and resulted in a 2 GB image. For AFF4-L, imaging took around 8 min and resulted in a 3 GB image. However, as mentioned in **Section 3**, modifications were made to a portion of the AFF4-L open-source code to image files larger than 1024 KB. Therefore, the imaging results for AFF4-L were obtained after making these modifications. It's worth noting that the current pyAFF4 open-source code available on GitHub does not support imaging files larger than 1024 KB. Additionally, for duplicate removal imaging, an error occurred in the code when the number of files exceeded 13, making it impossible to verify the results.

To ensure no issues arose when imaging large files, imaging was conducted for the 124 GB file named 'test_bigmulti.aff4'. Imaging with L01 took approximately 1 h and 28 min, resulting in a 106 GB image. For AFF4-L, imaging took around 50 min, resulting in a 116 GB image. However, when attempting duplicate removal imaging with AFF4-L, the process was not complete even after 5 days. Considering this impractical delay, it was determined that imaging with AFF4-L for this scenario was not feasible.

## 5. Results and discussion

The study aimed to address the three research questions proposed in the introduction. To achieve this goal, an analysis of the L01 and AFF4-L image file formats was conducted, identifying the metadata items that could be stored in each format. Subsequently, metadata manipulation tests and stress tests were performed. The comprehensive findings of this study are intended to provide answers to the research questions posed.

**Table 8**
Stress test results.

| Stress Test | | | 51,417 Files (18.9 GB) Imaging | 124 GB File Imaging |
|---|---|---|---|---|
| **L01** | | Result | O | O |
| | | Time | 12m | 1h 28m |
| | | Size | 2 GB | 106 GB |
| **AFF4-L** | **Logical Imaging** | Result | O | △ |
| | | Time | 8m | 50m |
| | | Size | 3 GB | 116 GB |
| | **Hash Based Logical Imaging** | Result | X | X |
| | | Detail | No more than 13 files | Over 5 days |

**RQ1: Are the existing logical evidence formats adequately suited for the selective acquisition of potential digital evidence from various sources?**

Existing logical evidence formats may not be sufficiently suitable for effectively managing potential digital evidence from various sources. A study targeting the most commonly used logical image format, L01, and the open-source format proposed in academia, AFF4-L, revealed that while L01 shares similarities with E01 and most fields can be identified, there are numerous unidentified fields and category identifiers in the detailed data. Moreover, despite containing a significant amount of metadata information, it did not encompass all such information when imaging various files from diverse file systems. Similarly, AFF4-L offered very limited metadata compared to L01 and provided only the same metadata information regardless of imaging various files from different file systems. Therefore, existing logical image formats do not contain sufficient metadata information necessary for effectively managing digital evidence from various sources.

**RQ2: Are there any weaknesses or deficiencies from an evidence management perspective?**

A detailed analysis of the internal structure of existing logical image formats demonstrated that they are being utilized in the process of logical imaging without sufficient consideration of how to preserve the originality and integrity of digital evidence. Due to the characteristics of the DEB format, interpreting and normalizing original metadata before storing it leads to a loss of originality. Existing logical evidence formats normalize the original metadata according to the format's predefined metadata structure. This approach makes it challenging to flexibly handle metadata from various sources and to fully preserve the original metadata. It was observed that both L01 and AFF4-L formats do not sufficiently reflect file system types or timestamp metadata. Furthermore, the metadata manipulation test results indicated that both L01 and AFF4-L formats were susceptible to metadata manipulation when encryption was not used. They lacked representative verification values for the image files themselves, and due to their simple and systematic format structures, metadata manipulation was relatively easy. Consequently, the tools supporting these formats failed to detect manipulations, leading to a potential failure of integrity verification. Due to the large amount of metadata stored in L01, selective metadata manipulation tests were conducted by prioritizing high-importance metadata for digital forensic analysis. The results showed that EnCase did not detect most of the manipulated metadata, with warnings appearing only for simple manipulation, while EnCase was able to confirm manipulated documents without warnings after elaborate manipulation. In the case of AFF4-L, the format verification was possible using pyAFF4 along with compression programs for handling the ZIP file format. Tests were conducted to determine if these tools could detect manipulated metadata. Simple manipulation was used to manipulate both image file metadata and original file metadata, with manipulated content confirmed through open-source tools. However, content verification using compression programs was not possible, but elaborate manipulation allowed confirmation of manipulated content even in compression programs through CRC manipulation. Both L01 and AFF4-L lacked representative verification values for the image files themselves, requiring investigative agencies to use separate tools for calculating hash values for generated image files. Therefore, it is necessary to consider providing additional verification information at the format level to confirm the manipulation of evidence and to ensure systematic evidence management.

Additionally, the stress test results revealed limitations in the performance of the pyAFF4 open-source tool supporting AFF4-L. Errors occurred when imaging files larger than 1025 KB, and issues were

encountered with open-source tools not functioning properly during deduplication imaging of large and numerous files.

**RQ3: If any limitations exist, how can they be improved for supporting thorough logical imaging?**

We propose solutions to address the identified limitations. For the loss-of-originality threshold, it is necessary to consider storing raw data alongside the normalized metadata provided by existing logical image formats. This means that the metadata storage structure of each file system, such as $MFT Entry for NTFS and Directory Entry for FAT, should be included as raw data to preserve its originality. This approach not only preserves originality but also allows you to interpret the raw data stored together and utilize all of the metadata if necessary.

To address the issue of integrity verification failure due to metadata manipulation, a solution could involve using digital signatures to certify the hash value of the image file with a public certificate. This involves encrypting the hash value with a private key to generate a digital signature, which is then stored in the form of a public-key certificate issued by a trusted third-party institution. To verify the validity of the digital signature in the future, the public key included in the public-key certificate can be used to decrypt and verify the digital signature. Through this method, image files can be transmitted while preserving their hash values, and the integrity of the file contents can be verified using the decrypted hash values, thus ensuring that the digital signatures were generated from the original files and that the files have not been altered in transit.

Finally, the overall limitations of existing logical evidence formats, such as the performance limitations of AFF4-L, suggest the need for continued development and standardization of logical image formats. AFF4-L open-source maintenance ceased after 2021. Although there have been continuous pull requests on the AFF4-L open-source GitHub repository since 2021, they have not been incorporated. Therefore, continuous development is necessary to effectively manage potential digital evidence from various sources using logical image formats. Standardization is also essential to enhance the efficiency and consistency of digital evidence management and analysis.

## 6. Conclusions and perspectives

Logical imaging has become increasingly important due to the growing demand for large-capacity storage devices, diverse computing environments, and the need for selective collection for privacy protection. However, there is a lack of in-depth discussion and research on existing logical image formats. Therefore, this paper raises this issue and further analyzes and manipulates existing logical image formats to identify their shortcomings.

Firstly, this paper identified the metadata of L01 and AFF4-L. Through this process, it successfully identified field and category identifiers in L01 that had not been previously recognized and highlighted the metadata deficiency issue in AFF4-L. Additionally, the study highlighted the issue of loss of originality due to the limited types of metadata in existing logical image formats and proposed storing raw data alongside to mitigate this problem. Metadata manipulation tests identified limitations related to the failure of integrity verification of the formats, and stress tests identified limitations in terms of the performance and reliability of the tools that support each format. Both L01 and AFF4-L showed vulnerabilities in easily manipulation with key metadata information such as hash values or timestamps when analyzing the format structure. Proposed improvements include providing verification information for image files themselves at the format level and suggesting hash value signing and verification using electronic signatures. Furthermore, for AFF4-L, it identified performance limitations, such as the inability to image files larger than 1025 KB using the currently available open-source version and the inability to perform imaging of large files and many files when using the deduplication option.

For future development, it is deemed necessary to maintain and update the AFF4-L open-source or develop a new format to effectively manage potential digital evidence from various sources. Furthermore, considering the increasing importance and significance of selective collection, there is a need for standardization of logical imaging through sustained interest and efforts from the entire community. Through these efforts, it is expected that logical imaging formats will align with current diverse environments and requirements, providing high efficiency and interoperability.

## Acknowledgements

This work was supported by a Korea University Grant, and also supported by Police-Lab 2.0 Program(www.kipot.or.kr) funded by the Ministry of Science and ICT(MSIT, Korea) & Korean National Police Agency(KNPA, Korea) [Project Name: Research on Data Acquisition and Analysis for Counter Anti-Forensics/Project Number: 210121M07].

## References

Cohen, M., Schatz, B., 2010. Hash based disk imaging using aff4. Digit. Invest. 7, S121–S128. https://doi.org/10.1016/j.diin.2010.05.015. https://www.sciencedirect.com/science/article/pii/S1742287610000423. the Proceedings of the Tenth Annual DFRWS Conference.

Bang, J., Yoo, B., Lee, S., 2011. Analysis of changes in file time attributes with file manipulation. Digit. Invest. 7, 135–144.. https://doi.org/10.1016/j.diin.2010.12.001. https://www.sciencedirect.com/science/article/pii/S1742287610000824.

Cohen, M., 2019. pyaff4. https://github.com/aff4/pyaff4.

Cohen, M., Garfinkel, S., Schatz, B., 2009. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow. Digit. Invest. 6, S57–S68. https://doi.org/10.1016/j.diin.2009.06.010. https://www.sciencedirect.com/science/article/pii/S1742287609000401. the Proceedings of the Ninth Annual DFRWS Conference.

EWF Family, 2015. Expert witness disk image format (ewf) family, 2024-05-19. https://www.loc.gov/preservation/digital/formats/fdd/fdd000406.shtml.

Faust, F., Thierry, A., Müller, T., Freiling, F., 2021. Selective imaging of file system data on live systems. Forensic Sci. Int.: Digit. Invest. 36, 301115 https://doi.org/10.1016/j.fsidi.2021.301115. https://www.sciencedirect.com/science/article/pii/S2666281721000093. dFRWS 2021 EU - Selected Papers and Extended Abstracts of the Eighth Annual DFRWS Europe Conference.

Garfinkel, S.L., Malan, D.J., Dubec, K.A., Stevens, C.C., Pham, C., 2006. Disk imaging with the advanced forensic format , library and tools. https://api.semanticscholar.org/CorpusID:10658527.

Halboob, W., Mahmod, R., Udzir, N.I., Abdullah, M.T., 2015. Privacy levels for computer forensics: toward a more efficient privacy-preserving investigation. In: FNC/MobiSPC. URL: https://api.semanticscholar.org/CorpusID:43688858.

Halboob, W., Almuhtadi, J., 2023. Computer forensics framework for efficient and lawful privacy-preserved investigation. Comput. Syst. Sci. Eng. 45, 2071–2092. https://doi.org/10.32604/csse.2023.024110. http://www.techscience.com/csse/v45n2/50368.

Han, J., Han, M.L., Lee, S., Park, J., 2024. Eco-bag: an elastic container based on merkle tree as a universal digital evidence bag. Forensic Sci. Int.: Digit. Invest. 49, 301725 https://doi.org/10.1016/j.fsidi.2024.301725. https://www.sciencedirect.com/science/article/pii/S2666281724000404.

Im, S., 2024. Dataset. https://github.com/ggeng2/Logical_Image_DataSet.

Knight, G., 2011. Forensic disk imaging report. Technical report. In: Forensic Investigation of Digital Objects (FIDO). The Report Examines the Principles and Practices Associated with Forensic Disk Imaging. https://researchonline.lshtm.ac.uk/id/eprint/354890/.

Magnet Forensics, 2020. Aff4-l support, portable case updates and more in magnet axiom 4.5 magnet axiom cyber 4.5. https://www.magnetforensics.com/blog/aff4-l-support-portable-case-updates-and-more-in-magnet-axiom-4-5/, 2024-05-19.

Metz, J., 2019a. Ewf specification. https://github.com/libyal/libewf/blob/main/documentation/ExpertWitnessCompressionFormat(EWF).asciidoc, 2024-05-19.

Metz, J., 2019b. Libewf. https://github.com/libyal/libewf, 2024-05-19.

OpenText, 2022. Encase forensic 22.3 release notes. https://cyber.quality-net.co.jp/webkanri/cbqualitynet27/wp-content/uploads/2022/08/EnCase_Forensic_22.3_Release_Notes.pdf, 2024-05-19.

Schatz, B.L., 2019. Aff4-l: a scalable open logical evidence container. Digit. Invest. 29, S143–S149. https://doi.org/10.1016/j.diin.2019.04.016. https://www.sciencedirect.com/science/article/pii/S1742287619301653.

Stüttgen, J., Dewald, A., Freiling, F.C., 2013. Selective imaging revisited. In: 2013 Seventh International Conference on IT Security Incident Management and IT Forensics, pp. 45–58. https://doi.org/10.1109/IMF.2013.16.

Turner, P., 2005. Unification of digital evidence from disparate sources (digital evidence bags). Digit. Invest. 2, 223–228. https://doi.org/10.1016/j.diin.2005.07.001. https://www.sciencedirect.com/science/article/pii/S1742287605000575.
Turner, P., 2006. Selective and intelligent imaging using digital evidence bags. Digit. Invest. 3, 59–64. https://doi.org/10.1016/j.diin.2006.06.003. https://www.sciencedirect.com/science/article/pii/S174228760600065X. the Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).
Velocidex, 2019. winpmem_v3.3.rc3.exe. https://github.com/Velocidex/c-aff4/releases, 2024-05-19.