Applying digital stratigraphy to the problem of recycled storage media

Janine Schneider, Maximilian Eichhorn, Lisa Marie Dreier, Christopher Hargreaves

Angaben zur Veröffentlichung / Publication details:

Schneider, Janine, Maximilian Eichhorn, Lisa Marie Dreier, and Christopher Hargreaves. 2024. "Applying digital stratigraphy to the problem of recycled storage media." *Forensic Science International: Digital Investigation* 49 (Supplement): 301761. https://doi.org/10.1016/j.fsidi.2024.301761.

Forensic Science International: Digital Investigation 49 (2024) 301761

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS USA 2024 - Selected Papers from the 24th Annual Digital Forensics Research Conference USA

Applying digital stratigraphy to the problem of recycled storage media

Janine Schneider [a,b,*], Maximilian Eichhorn [b], Lisa Marie Dreier [b], Christopher Hargreaves [c,**]

[a] CISPA Helmholtz Center for Information Security, Germany
[b] Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
[c] University of Oxford, United Kingdom

ARTICLE INFO

Keywords:
Digital stratigraphy
Digital forensics
File systems
Simulation
Removable storage

ABSTRACT

Previous work has shown that second-hand or even new devices with recycled components can contain remnants of old data. Given a situation where incriminating evidence is found in non-allocated space of such a device, this presents an attribution problem. In archaeology or geology, stratigraphy studies the arrangement of strata, or layers, often used as a dating technique based on the premise that newer layers are situated above older layers. The digital stratigraphy technique applies the concept to digital forensics and considers how data is positioned and overlayed on disk to make inferences about when data was created. This research investigates the extent to which this technique could resolve the data provenance challenge associated with recycled digital storage media. This paper presents an automated file system activity simulation framework that allows creation, deletion and modification actions to be carried out at scale using specific file system drivers. Using this tool, a series of experiments are carried out to gain an understanding of file system driver behaviour and address this practical question of provenance of data in non-allocated space.

1. Introduction

Insufficient data sanitisation practices can complicate digital forensics. The persistence of data from previous use of the storage media may raise uncertainties about whether the current owner is knowingly in possession of potentially incriminating material. This problem has been shown to exist with second-hand storage devices (Garfinkel and Shelat, 2003; Freiling et al., 2008) and recent work has shown that this problem also exists for newly purchased USB devices, which may contain residual old data due to recycled internal components (Schneider et al., 2021). This raises important attribution challenges to all data found in non-allocated space on USB storage media, regardless of whether it was second-hand or newly purchased. It is therefore important to conduct additional work to determine in what circumstances data on such devices could be used as evidence, specifically, in which circumstances content recovered from non-allocated space could be used as evidence.

This paper addresses this problem using digital stratigraphy techniques, and specifically considers: *if incriminating recovered content is identified in non-allocated space on removable storage media that potentially contains old data, is it possible to differentiate whether that data was deleted from the current file system or potentially was already present?*

1.1. Existing approaches

In order to be able to use a deleted file as evidence, the file must first be reconstructed. This reconstruction can be performed based on various properties of the file (Pal and Memon, 2009) and can be non-trivial. Casey et al. (2019) describes different formal classifications of file recovery, highlighting the imprecision in language and within tools regarding the reliability of recovered data from non-allocated space. This is highly relevant to this problem since data in non-allocated space on devices that are recycled or have recycled components should be in one of their well-defined states. These are not repeated in full here, but files are classed as having three distinct conceptual parts, each of which may be recoverable or not (in different combinations): filename, file metadata, and file content.

Assuming some content of interest is found on USB storage media (with a potentially recycled memory chip) in non-allocated space, a potential defence could be that since the file was not part of the current file system, it could have already been present when the device was

* Corresponding author. CISPA Helmholtz Center for Information Security, Germany.
** Corresponding author.
E-mail addresses: janine.schneider@cispa.de (J. Schneider), maximilian.eichhorn@fau.de (M. Eichhorn), lisa.dreier@fau.de (L.M. Dreier), christopher.hargreaves@cs.ox.ac.uk (C. Hargreaves).

https://doi.org/10.1016/j.fsidi.2024.301761

Available online 5 July 2024

purchased. Depending on the specific recoverable data, several approaches could be used to investigate the validity of this defence. In the following subsections, content recovery is assumed, and filename recovery is grouped with metadata recovery for brevity.

1.1.1. Recoverable metadata

First, if a metadata structure can be found in the current file system (e.g. a deleted MFT entry) that includes a reference to the recovered content, then it can be argued that this file came from the current file system and, therefore, potentially the suspect in question (although attempts to link to a suspect are not covered here, only links to the current file system of the device).

It might also be possible to locate an entire metadata structure that is in non-allocated space and not directly linked to the current file system, e.g. an entire FAT32 directory entry structure. In this case, there are multiple options as this could either be because it came from an older file system, or it has become detached from the current file system in some way; perhaps an entire directory was deleted and the reference to that directory overwritten. On some file systems, there may be approaches to link this to something in possession of the suspect, e.g. in Nordvik et al. (2019), it is discussed that the Birth Volume Object ID on NTFS may be used to link a file to the volume it first obtained an ID, or obtaining MAC addresses from ObjectIDs.

However, if there is evidence of an older file system, if the metadata structure of the incriminating content is of a different type, i.e. a FAT directory entry found when the active file system is NTFS, then this could be from an earlier file system. However, this does not apply if the current and previous file systems are the same type. There is also some further complexity here regarding file system container file formats, for example, a file that is a virtual machine disk (as also discussed in Casey et al. (2019)), as this would be a valid reason for metadata structures from a different file system to be present.

1.1.2. No file metadata

If no file metadata structures that refer to the recovered content are available, there are still some approaches to try and link this data to the current file system. In some circumstances, the file may contain internal metadata (Bahjat and Jones, 2019) suggesting it is from the current file system. This could include internal timestamps within the lifetime of the current file system or a reference to the suspect's username. It could also include EXIF data of a camera that is either in the possession of the suspect or the suspect has pictures on other devices taken with the same camera. In these cases, attempts can be made to link these pieces of metadata with the suspect.

Alternatively, it may be possible to find details that suggest that the file was not associated with the current file system, e.g. if a recovered file is sector-aligned but is not aligned with the cluster boundaries of the current file system, in this case, this could indicate that this file did not come from the current file system, although file system containers could again explain this. It may also be possible to use data from another device to demonstrate a link to the deleted content, e.g. link files on a Windows system referring to files on removable media (Patterson and Hargreaves, 2012), or a file has internal metadata that has a matching name with a link file on the suspect's device.

1.1.3. The worst case?

Finally, if some recovered content is of significant interest, has no file metadata structures, has no internal metadata linking it to another system or device of the suspect, and there is no contextual information within the file content itself, then this data of interest is very difficult to attribute to the current file system since it cannot be shown that this data was deleted from the current file system.

1.2. Research goals, contributions and paper structure

The aim of this work is to investigate this remaining challenging situation, to either demonstrate that carved content could originate from a previous file system (defence perspective), or that it could be part of the current file system and has been deleted (prosecution perspective). This is achieved through the use of digital stratigraphy, and the study of how two file systems interact when one is written on top of the other. This work makes the following contributions:

- It provides substantial experimental data that shows when carved content could, or could not have been part of the current file system (considering FAT32, NTFS, and exFAT on Windows and Linux).
- It provides a general method and open-source automated File System Activity Simulator for conducting experiments with file system drivers on different platforms at scale to understand their behaviour.
- It provides new tools and visualisations to better understand the write patterns of file system drivers, and tools to examine a disk image and analyse carved content in the context of other file system data.

The remainder of the paper is structured as follows: Section 2 reviews existing work on digital stratigraphy, and Section 3 provides the overall methodology for this work. Section 4 describes the automation framework that was developed, and Section 5 summarises the results of the experiments conducted. Section 6 reviews the research questions in light of the results, and a demonstration of the concepts presented is provided in Section 7. Section 8 provides a discussion of the limitations and further work, followed by conclusions in Section 9.

2. Related work

2.1. Stratigraphy in archaeology and geology

The origins of stratigraphy are in the disciplines of archaeology (Harris (1987)) and geology (Lyell (1872)). In both disciplines, the fundamental idea is to recognise 'layers' of rock or sediment and to establish a succession that can be interpreted to understand how it evolved. This is based on rules that are derived from observations and natural laws, e.g., the observation that new sediment accumulates on top of older sediment; thus, a stratigraphic unit lying below another one typically was created earlier and thus is assumed to be older Harris (1987).

2.2. Digital stratigraphy

These concepts were transferred to digital forensics in Casey (2004) and formalised as digital stratigraphy in Casey (2018), which investigates whether knowing the allocation behaviour of a file system driver allows certain information about the creation order of files to be derived from their positions on disk.

Other related work includes Willassen (2008), which investigated MFT and journal entries as file allocation traces. Employing the sequence numbers in the file system's journal file and formal reasoning about the allocation sequence (based on the first-fit allocation strategy together with a counter for entry-reuse), it is shown how to establish the relative ordering of the MFT entries and thus find traces of file antedating.

Others have investigated FAT32 file system drivers through source code or dynamic analysis. Tse (2011) concludes that the position of the first cluster of a file and the average of all cluster locations of a file are good metrics to determine the relative ordering of files, at least as long as the file system is not heavily fragmented. Hargreaves (2013) also provides a visualisation of block allocation within digital forensics, essentially providing a two-layer stratigraphy view.

Minnaard (2014) used static code analysis of a Linux FAT32 driver as well as dynamic analysis of a Windows one to find differences in their behaviour. For the Linux driver, Minnaard (2014) proposes that the creation order of the files can be directly derived from the positional

order of the files, which is disputed in Lee et al. (2015). They point out that wrap-arounds destroy this direct mapping and add that each wrap-around creates files and thus creates one lower and one upper bound for the creation time of a certain file. Still, they admit that they did not find a way to determine the candidate file for those bounds. Additionally to the above mentioned, Li et al. (2016) analysed the FAT32 drivers for Windows and Linux. They dived slightly deeper into the technical description than Minnaard (2014), but did not try to infer information about the temporal relationship between creation dates of files.

Even though Karresand et al. (2020b) aimed to improve file carving and does not strictly fall in the category of digital stratigraphy, but it does investigate the allocation strategy of NTFS drivers (Karresand et al. (2020b); Karresand et al. (2020a); Karresand et al. (2019); Karresand (2023)). Thus, many of the findings are relevant for digital stratigraphy as well, e.g., the result that allocation behaviour changed between Windows 7 and Windows 10, that Windows deviates significantly from a best-fit strategy or that the disk does not seem to influence the allocation strategy (Karresand et al. (2020b)).

Previous work has not only investigated allocation behaviour to derive relative ordering, but also to approximate the absolute date using digital stratigraphy. This is the aim of two studies conducted by Bahjat and Jones (2019) and Bahjat and Jones (2023). Both studies take the creation time of the k-nearest neighbours of a file fragment and try to approximate the file fragment's creation time using a lower and an upper bound.

Vollebregt (2019) takes a similar approach, estimating the lower bound of the creation time as well as the upper bound of the deletion time of a file. The approach assumed that the Windows 10 NTFS file system driver uses a best-fit-strategy, but found that the NTFS file system drivers used in their real-world datasets differ heavily from that behaviour. There is also recent work presented in Bojic et al. (2022), which showed the potential of simulating file system usage and recording successive states to observe and understand file system driver behaviour. Other work examines changes in file systems over time for use in digital forensics, including visualisations. For example, Hargreaves and Chivers (2010) describes a method for detecting hidden volumes within encrypted containers if multiple copies can be acquired, for example, from Volume Shadow Copy. Visualisations of the changes over time are shown, and information about the size of the hidden container can even be inferred.

Therefore, there is a reasonable set of work attempting to date files, absolutely or relatively. None have focused on the specific challenge of recycled removable media, but the existence of this work and basing techniques upon this knowledge base shows the potential for the approach.

3. Methodology

3.1. Aim

The focus of this work is to investigate if it is possible to use digital stratigraphy techniques to provide any attribution information to 'content only' data found in non-allocated space on a device that potentially uses recycled storage components, and link that content to the current file system.

It is hypothesised that live data will be written in an incremental manner moving through the blocks of the storage media. Therefore, we also assume that there is something that could be referred to as a File System Upper Bound (FSUB), which is *the largest block number in which data from the current file system is known to have been written*. This is discussed in Section 5.4.

Assuming that this is applicable (which will be evaluated later), there are two scenarios for identified relevant content: First, shown in Fig. 1a), relevant data is recovered from non-allocated space before the FSUB of the current file system. Second, shown in Fig. 1b), relevant data is recovered from non-allocated space beyond the FSUB of the current file system. Therefore this leads to several research questions:

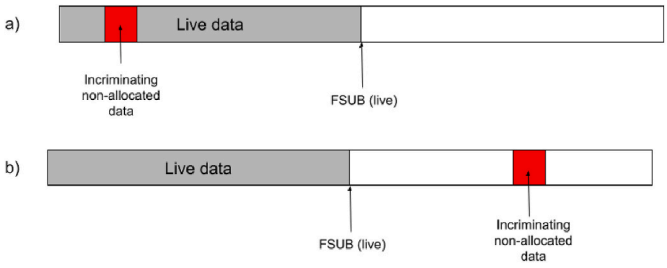


Fig. 1. Shows different scenarios where incriminating evidence is located in different positions relative to the current File System Upper Bound.

1. How can it be determined if data is written in a sequential manner for a given file system?
2. How can the File System Upper Bounds (FSUB) be determined?
3. If data of interest is found in non-allocated space, but prior to the FSUB of the current file system, could this result from poor sanitisation practises and, therefore, not be associated with the current file system?
4. If data is found further into the disk than the current file system FSUB, is it possible that it was part of the current file system?

3.2. Overall method

The method chosen is to perform file system activity simulations to understand the file system driver behaviour on multiple systems. To achieve this, an automation and visualisation framework is developed that is discussed in Section 4. A series of experiments are conducted that investigate these research questions, which are explained and documented in Section 5.

4. Automation framework development

This section describes the development of the 'File System Activity Simulator' (FSAS) used to support the experiments in this paper. It is designed to create file system states that are representative of real-world systems and to programmatically conduct operations on the file system using regular OS interfaces. Importantly, it also preserves the file system's state after each operation and visualises the results. The Python script *fsas.py* is implemented for this purpose. This consists of three separate commands, which are explained below.

4.1. Generating file system operations

The first command, *generate*, creates a flowchart for the operations in a file system and stores it as an XML file. This flow chart generation has a random element, but the result is static. This mirrors the process of generating forensic disk image generators in Moch and Freiling (2009). This allows the use of randomness but in a repeatable manner.

Each created XML file can be executed as often as desired on Windows and Linux systems. Support for other operating systems can be added at any time thanks to the modular structure of the FSAS. The respective parameters can be taken from the help text and are checked when the script is executed. The script offers the file operations *create*, *shrink*, *enlarge*, and *delete*, as well as the directory operations create and delete, to simulate as many application scenarios as possible. After defining the available operations, the individual operations' parameters are randomly selected in a loop based on the specified framework conditions. This procedure is subject to several consistency checks. For example, no file can be deleted, reduced or enlarged if no file was created beforehand. If all checks are passed, the XML file can be written.

4.2. Executing file system operations

The second command, *execute*, calls a Python function that executes the previously created XML file on a specific file system. Various parameters can be selected for this script, for example optional parameters can be used to select different output types that preserve the state of the file system before the first operation and after each operation. These outputs include various additional tools that must be accessible via the PATH environment variable when used. In addition, the specially developed output formats *alloc*, *nonzero*, and *pattern* can be selected. However, these use *The Sleuth Kit* (TSK) Python bindings *pytsk* and can only be used for file systems whose support is implemented in the bindings. The script reads, checks and executes commands from the XML. Before the first operation is performed, the selected tools are applied to the initial file system. After each operation, an output is created and logged in the output directory using the enabled tools.

4.3. Generating visualisations

Finally, *plot* forms the third command. The different plot types *casey*, *scatter*, and *heatmap* can be selected for this command (all discussed later). At this point, several auxiliary files are created according to the plot type, which can be very large depending on the file system size being examined. For the creation of plots of type *casey* and *scatter* the Python library Matplotlib is used, and for the *heatmap* Plotly is used.

5. Experimental results

5.1. Overview

This section provides the experimental results used to address the research questions. First, file system driver behaviour is explored in Section 5.2, highlighting allocation behaviour and differences between Windows and Linux drivers. Section 5.3 summarises additional findings regarding file system specifics on different operating systems.[1]

Section 5.4 then investigates the concept of File System Upper Bound theorised earlier, showing how it can be calculated and providing tooling for it. Section 5.5 then conducts experiments considering if data from old file systems can be found in the current file system prior to the FSUB. These results focus on the file system behaviour, but acknowledging the partitioning in which file systems sit, in addition experiments were performed with formatting the removable storage media. These are not included in full due to space constraints, but in summary, the experiments show that 'quick format' does not zero-fill the file system, in contrast to the non-quick format option. Furthermore, space outside of the partition remains untouched.

5.2. File system driver behaviour

This section discusses experiments investigating the basic patterns of file system driver behaviour across different driver versions and operating systems.[2]

Casey (2018) provides plots for digital stratigraphy from a disk image using the first cluster on the y-axis, and created timestamp on the x-axis. As discussed earlier, the FSAS is able to output plots in this style. The benefits and limitations of this plot type will be shown in this section, along with additional plot types to complement this original plot type. The remainder of this section describes the specific findings from the experiments.

5.2.1. Sequential writes

The term sequential writes refers to an allocation behaviour where blocks are allocated in an incremental manner forming linear allocation patterns. To investigate whether FAT32, exFAT or NTFS show such a behaviour, experiments are conducted where the formatted partition is populated with files until 95 % of the file system space is in use. Afterwards, a *casey* plot (Casey, 2018) is created for each of the resulting final images. For FAT32 and exFAT, sequential allocation behaviour for these experiments could be confirmed on Linux and Windows. However, a different behaviour could be observed for NTFS as there are areas with sequential allocation on both platforms, but these are not contiguous (see Fig. 2). On Windows, the areas of sequential allocation are much broader. Meanwhile, with Linux, these areas are interrupted by areas in which high fluctuations can be observed.

5.2.2. File deletion and block reuse

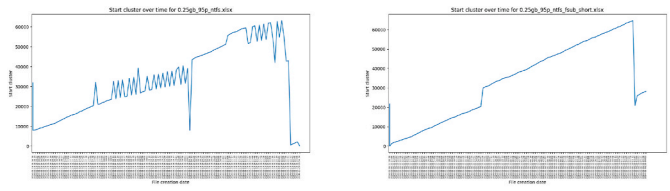
Additional experiments investigated how the different file systems behave when deletion is involved. This time the automated FSAS created and deleted files randomly until 80 % of the file system space was in use.

Furthermore, a new visualisation was developed to analyse this set of experiments, allowing analysis of the status of blocks. This visualisation differs from the visualisation in Casey (2018) as a scatter plot is used instead of a line plot, and block groups of four blocks each are used instead of individual starting clusters. Multiple variations of the plot are possible e.g. one showing the allocation status (blue) and one showing where known file content can be observed (green). Only the allocation status plots are shown for brevity. In addition, the axes are reversed, which corresponds to archaeology stratigraphy representations, and is important later for consistency as other visualisations are added, which show data being written on top of other data.

Since each individual block status in the file system is to be visualised across many operations, data reduction is essential. Therefore, the visualisation shows only the changes of each block and, thus, implicitly the status after each operation. The first line of each scatter plot shows the status of the file system before the first operation, all blocks (except those used to store file system structures) are considered not allocated and not containing file data.

The results of the experiments show that when file deletion is involved, there are differences between the file and operating systems. With FAT32 on Linux, files are still written to incremental start clusters, even after deletions occur, and any de-allocated blocks are not immediately reused. Only after the end of the file system is reached, are the open gaps filled again sequentially, provided they are large enough. On Windows, any de-allocated blocks (from deletions) are filled again if they are wide enough, as seen in Fig. 3.

The immediate filling of gaps can also be observed with exFAT. However, there are no noticeable differences between the exFAT allocation patterns on different operating systems. Regardless of the differences already mentioned for NTFS, smaller platform-related



(a) Linux (b) Windows

Fig. 2. NTFS file system allocation behaviour on Linux (left) and Windows (right) shown in a Casey-style plot. In this experiment, 95 % of a 250 MB file system space was allocated.

[1] Please note that both sectors and clusters/blocks are referred to in this section. However, because of the underlying pytsk implementation of many of the tools, for FAT32 and exFAT, only sectors are reported since the library does not provide cluster values for those file systems.

[2] The OS driver versions used in these experiments are: Debian 11 dosfstools 4.2, ntfs-3g v2017.3.23AR.3, exfatprogs 1.1.0 and Windows 11 FAT 10.0.22621.2506, NTFS 10.0.22621.2715, exFAT 10.0.22621.2506.

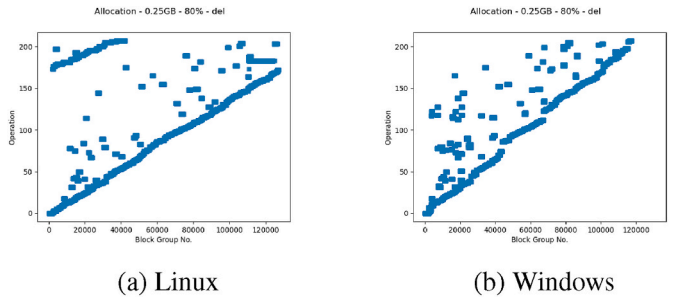


Fig. 3. FAT32 file system allocation and deletion behaviour on Linux (left) and Windows (right) (shows allocation status changes). In this experiment, 80 % of a 250 MB file system space was allocated in summary. Shows that on Linux the allocation only returns to old blocks once the end of the volume is reached, whereas Windows fills old blocks earlier.

differences can be identified when deleting files, as can be seen in Fig. 4. Gaps that occur with NTFS are also filled. Unfortunately, making more precise statements about this behaviour based on the plots is difficult.

However, additional file system specifics can be observed when looking at the plots and the underlying data more closely.

5.3. File system specifics

While analysing the results of the experiments on the file system driver behaviour, some file system specific features became apparent depending on the different operating systems, which are presented in the following.

In order to analyse this in more detail, the visualisation was developed further and evolved into a dynamic heatmap. The advantage of using a dynamic heatmap instead of a static scatter plot is that different colours can be used to visualise actual status (instead of implicit status) and combinations of statuses. Furthermore, interesting data locations can be zoomed in (in the HTML version of the plot). An example of the advantages can be seen in Figs. 5 and 6 which are equivalent alternative representations of Figs. 3 and 4. With the help of the heatmap colour visualisation, some interesting specifics can be observed for FAT32 and NTFS. The most important colours are: light green (allocated file data), dark green (non-allocated deleted file data), yellow (allocated but zero-filled), brown (content that is not allocated, non-zero and does not contain the standard file pattern, which sometimes can indicate file system metadata) and purple (old data, discussed later).

5.3.1. FAT32 specifics

The beginning of each partition formatted with FAT32 contains the two File Allocation Tables (FATS). However, the two FATs on Linux are larger than on Windows, and they also have different locations, most likely due to the different default cluster sizes for these volumes (512 bytes for Linux and 2048 bytes for Windows).

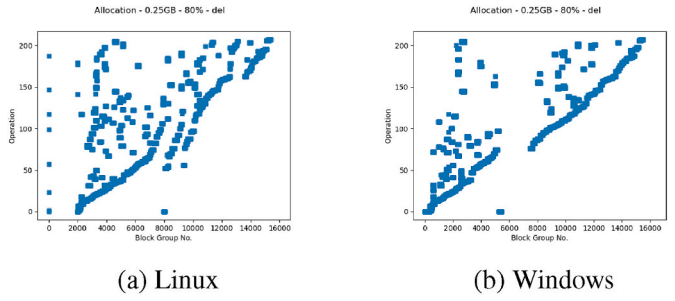


Fig. 4. NTFS file system allocation and deletion behaviour on Linux (left) and Windows (right). In this experiment, 80 % of a 250 MB file system space was allocated in summary. The diagram shows the allocation status changes of each block over the different operations.

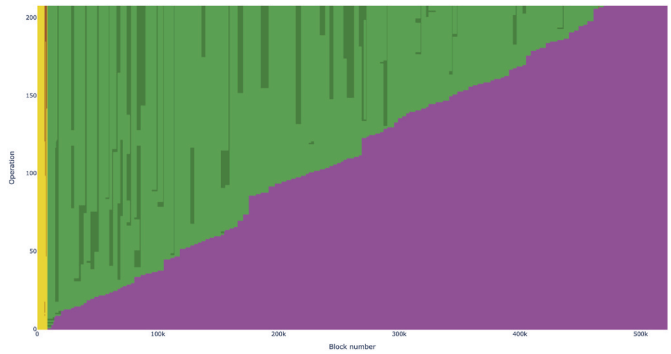


Fig. 5. FAT32 file system allocation and deletion behaviour on Windows. In this experiment, 80 % of a 250 MB file system space was allocated. Equivalent of Fig. 3.

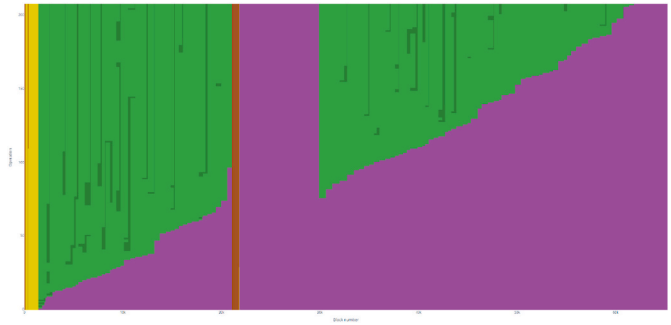


Fig. 6. NTFS file system allocation and deletion behaviour on Windows. In this experiment, 80 % of a 250 MB file system space was allocated. Equivalent of Fig. 4.

5.3.2. NTFS specifics

With NTFS under Linux, a reserved area can be observed at the beginning of the file system, where new file data is written only after this area. There is also a second reserved area initially marked as allocated and located deeper into the volume. The area at the beginning of the file system contains the $MFT and builds its reserved area, whereas the later area contains the $MFTMirr. On Windows, those two file system structures are switched.

Sequentially allocated areas are only occasionally found under Linux. Whereas on Windows, the file system is written sequentially up to and after the $MFT. This results in a gap encompassing the $MFT and its reserved area. Data is only written to this area if no more space is available.

5.4. File System Upper Bound (FSUB)

The data generated in the previous section can also be used to investigate the concept of a calculable FSUB, defined as *the largest block number in which data from the current file system is known to have been written*. The method for calculating the FSUB from a file system has subtle differences depending on the file system. There are also two FSUB values that can be identified, one calculated from the live files in a volume, and another using information from deleted metadata structures that can still be linked to the current file system.

For FAT32, with live files, the furthest block in use by the file system can be easily determined by iterating through the live files, following each of their FAT chains and recording the highest block in use by any live file (this can be seen on the right of Fig. 5). To include data from deleted files on FAT32, since the FAT itself is zeroed when a file is

deleted, the cluster number with the most confidence is the 'first cluster' recorded in the directory entry for a deleted file. However, this is potentially incomplete since a file can occupy more than one cluster if it is larger than the cluster size from the volume, so a 'possible FSUB' can be calculated using the start sector of the file furthest into the volume, plus the size of the file, although there is uncertainty in the accuracy of this since files can be fragmented.

For NTFS, it is possible to determine the highest block number in use by live files by considering each data run for each live MFT record. This approach can also be used on deleted files in NTFS since the data runs remain intact even after file deletion (until a new file overwrites the record). However, the complexity in NTFS, as seen in the previous section, is that specific metadata files ($MFT or $MFTMirr) are written part way into the volume. Therefore, the naive calculation of the FSUB is not always representative.

5.4.1. Validating FSUB concept

From the experiments conducted, it can be seen that data is written incrementally, in some cases moving back to fill de-allocated blocks, but it is possible to derive the highest block used. Since the aim is to use the FSUB position to determine if recovered content could have been part of the current file system, it is necessary to determine the FSUB not from the incremental experiments but from a disk image and to determine if this is representative.

To validate this concept, a tool (*get_fsub_info.py*) was written in Python that uses *pytsk* to identify and display the FSUB based on live files but also deleted. Experiments were performed that use the File System Activity Simulator initially only to create files and track the highest written block, and to compare it with the values calculated using *get_fsub_info.py* targeting the final disk image. This was effective, and all values corresponded. The second part of the experiment included multiple deleted file behaviours in the process and compared the highest block number written during the experiment with the calculated FSUB from the final resulting disk image. An extreme example is visualised in Fig. 7 and shows the creation of files up to 95 % of the disk, and then deleting the most recent files down to about 50 % capacity. A large difference between the FSUB from the disk image and the historically logged FSUB can be seen. For this example, the maximum recorded sector was 472,019, but the FSUB calculated from the approach in *get_fsub_info* recorded only 271,011. This shows some of the worst-case performance of this approach, although potential improvements are discussed in Section 8.

This shows that an FSUB can be calculated for a disk image, but there are limitations due to the inherent unreliability of deleted file recovery. Therefore, an FSUB calculated for a disk image is likely to be a *conservative lower estimate* of how far into the volume files have been written, and according to all data generated, it cannot be an overestimate.

5.5. Investigating previous file system data prior to FSUB

With the concept of an FSUB demonstrated and calculable, this section considers the possibility of data from a previous file system manifesting below the FSUB. In order to investigate this question, the disks used for the previously described experiments were initialised with a specific pattern ('this is old data') before the disks were formatted (as discussed earlier, leaving much of the partition filled with the old pattern). With this baseline established, experiments were conducted, file system operations were performed, and files were written with a different consistent pattern, meaning that new data can be differentiated from old data. Since the new visualisation allows the different states of blocks to be represented, it is possible to see when old data remains visible in specific disk areas. Furthermore, a tool *check_for_old_data.py* was developed to compliment the visualisations, which allowed a simple output of all blocks that contained the old data pattern.

5.5.1. FAT32

For Windows and Linux, the old data pattern was visible above the FSUB until the end of the volume. No old data was visible in the file system reserved area or the FATs in all resulting datasets. However, within the data area, several sectors containing old data were found in all experiments for Windows. This was investigated further, and it was found that the number of contiguous sectors containing old data was between 1 and the cluster size minus 1. This is because while sector slack of files was found to be zeroed, the sectors that contained old data were the cluster slack of new files written to the file system. Aside from these instances, no old data was found within the volume until the FSUB was reached. For Linux, the cluster slack of files was zeroed, and no old data was found before the FSUB.

5.5.2. NTFS

On Windows, some old data was found very early on in the volume, confined to file system files, for example, `$Extend/$RmMetadata/$Repair/$Corrupt`. In addition, as shown earlier in Fig. 6, since the $MFT is located part way into the volume, and because an area is initially reserved for expansion, old data can potentially be found in significant amounts prior to the FSUB. On Linux, old data was located at the start of the volume (typically cluster 3), but also in a large number of clusters representing the MFT reserved area, since for Linux, the $MFT was found at the start of the volume rather than later on. Also, since data is sometimes written on either side of the $MFTMirr file without first reaching it, there can be large areas of old data that can be found, as shown in Fig. 8.

Therefore, on NTFS, if old data was originally present on the disk, it is very possible to locate it within the FSUB of the new file system. On FAT, that is also true, however, the files are written contiguously (and may go back and fill de-allocated blocks), and so the old data that would potentially be found is limited to runs within the cluster slack of written

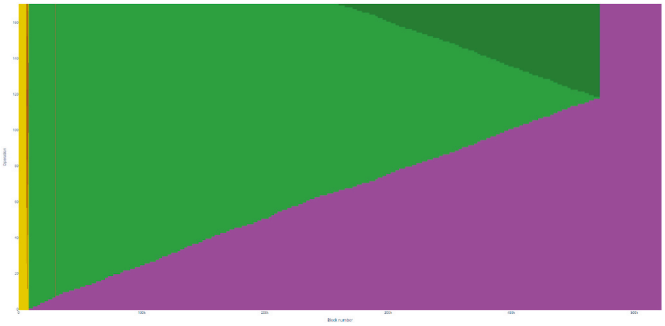


Fig. 7. This shows the addition of files on FAT32 followed by deleting the last 40 % of those created. You can see that the final FSUB derived from the live data would be less than the furthest block allocated over the life of the volume.

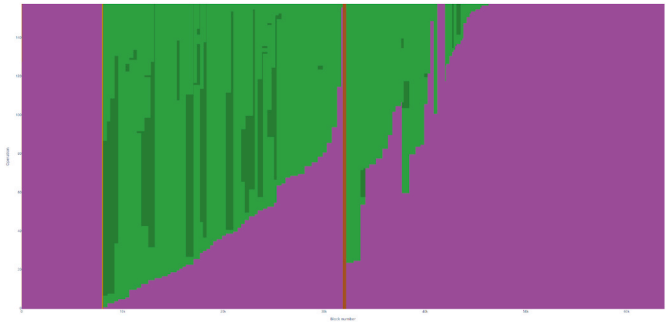


Fig. 8. Shows NTFS on Linux, the distribution of live data on either side of $MFTMirr, without necessarily reaching it, resulting in old data (purple) being found within the FSUB.

files and, therefore, limited to `cluster_size - 1 sector`.

5.5.3. exFAT

For exFAT on Windows, old data is found very early on in the volume in the reserved area, and in some system files (typically cluster slack of files such as $Upcase), and interestingly also in the FAT. Once in the data area, the properties regarding old data preservation are similar to FAT, and while areas of old data are present prior to the FSUB, they are exclusively in the cluster slack of new files written.

6. Discussion

In this work, over sixty experiments have been conducted, and only a small selection is possible to present here. Nevertheless, many interesting insights into file system driver behaviour have been gained. This section revisits the research questions proposed in Section 3.

Research Question 1 was concerned with determining if data is written in a sequential manner for a given file system, and it has been shown that for FAT and exFAT, this is generally the case. NTFS is more complicated due to file system metadata structures in the middle of the volume, and this has implications for other aspects of the work, which is discussed later.

Research Question 2 asked: how determining the File System Upper Bound could be achieved. It has been shown that it is possible to calculate a value for FAT, exFAT and NTFS by looking at both live and deleted files on a disk image and noting the blocks that they occupy. For file systems such as NTFS, because of the non-sequential allocation, the naïve FSUB calculation is not always representative because of the file system metadata structures part way through the volume. However, for the others, the naïve approach is effective. However, even then, uncertainty around deleted files makes any calculated value likely to be a conservative lower estimate of the true value. This is, however, in some ways advantageous. Since we are discussing the potential of using data from non-allocated space as evidence below the FSUB, the fact that this is a very conservative estimate by design should give increased confidence in results from below the FSUB.

Research Question 3 asked: if data of interest is found in non-allocated space, but prior to the FSUB of the current file system, could this be a result of poor sanitisation practises and, therefore, not associated with the current file system? For FAT, it has been shown that if data is located prior to the FSUB, and is larger than the available cluster slack of a file on the current file system, experiments have been unable to show that this data would be the result of poor sanitisation practises, and the data is more likely to have come from the current file system. This represents a significant improvement, given the potential implications of Schneider et al. (2021) regarding removable media. For exFAT, it is similar, but the data area only must be considered as old data may be found in the reserved area or FAT. However, for NTFS, more work is needed as it has been shown that there are multiple ways in which data could be located before the FSUB e.g. for low use drives the area between the $MFT (on Windows) and the highest allocated cluster of user added files (rather than NTFS metadata files). Further work refining the FSUB concept for more complex file systems may, in future, allow confidence in recoverability in some cases.

Research Question 4 asked: If data is found further into the disk than the current file system FSUB, is it possible that the data was part of the current file system? Experiments have shown that since calculating the FSUB from a disk image does have inherent uncertainty, it is possible that data above the FSUB could have been part of the current file system, especially if near to the FSUB or there have been a large number of recent deletions. However, it is currently difficult to prove this association when comparing the data to those below the FSUB, and given the experimental results to date, the data obtained above the FSUB should be treated with caution. Further work may be able to evolve the FSUB calculation, which is relatively simple at present, based only on block allocations of live files and recoverable deleted files with metadata, but other techniques, such as using files with linkable internal metadata, would allow the FSUB to be extended with more confidence, but further work is required.

7. Demonstration

With the knowledge gained in the previous section, it is possible to demonstrate an approach to address the original problem of data of interest being identified in non-allocated space from removable media (e.g. via carving).

If it is possible to observe the carved data in the context of the overall structure of the current file system, then it should be possible to reason about the relevance of that data.

In order to achieve this, a tool *df_digger.py* was developed. This tool does not use experimental data, but instead targets a disk image, for example from an investigation in which the context of some carved data needs to be understood.

This tool produces a comparable visualisation to the experiment data, with block numbers of the volume on the x-axis, but on the y-axis, as disk images do not have incremental full records of changes to the file system, this is instead inferred from file metadata. Each file on the file system is considered in sequential order according to their created timestamp. In addition, the blocks occupied by that file are determined, and the tool treats each file creation as equivalent to an experimental 'operation' and creates a similar visualisation, showing incremental allocation of blocks.

Figs. 9 and 10 show plots for FAT32 and NTFS, generated entirely from final disk images. If these are compared with Figs. 5 and 6 earlier, the structure and trends can be seen to be approximately the same, but without detailed information about file operations, such as what has been deleted from where. However, the FSUB can be clearly seen for FAT32, in addition to the patterns of writes as to how that FSUB was reached, and for NTFS, the complexity of the allocation algorithms and MFT reserved space can be seen.

To show data recovered from non-allocated space in the context of other file system data, *df_digger.py* allows additional parameters (–carved_start, –carved_end) to represent some carved data of interest in the top rows of the visualisation.

With the knowledge gained through the experiments and this new visualisation tool, it is now possible to return to the original problem. Two disk images representing two scenarios were manually created with no automation, and no manual disk edits. Scenario one recreates the case where incriminating data was stored on the disk prior to the suspect receiving the device. A file of interest was carved from sectors late in the disk. Fig. 11 shows *df_digger's* representation of that carved data in context, where the identified content of interest is shown to be beyond the FSUB, and there is no evidence of any data from the current file system being written in that area. In this case, use of that carved data as evidence would be unwise.

Scenario two recreates the case where the suspect stored

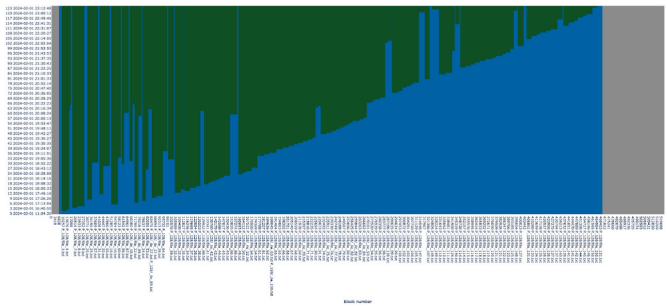


Fig. 9. Shows a visualisation of a FAT32 disk image (Windows) with data derived from the final disk image. Green is allocated data, and blue is non-allocated.

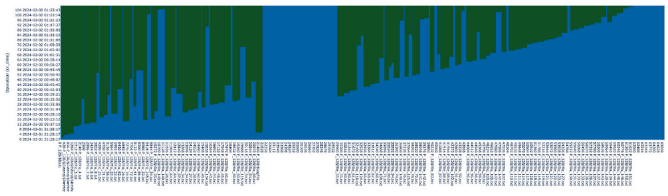


Fig. 10. Shows a visualisation of a NTFS disk image (Windows) with data derived from the final disk image. Green is allocated data, and blue is non-allocated.

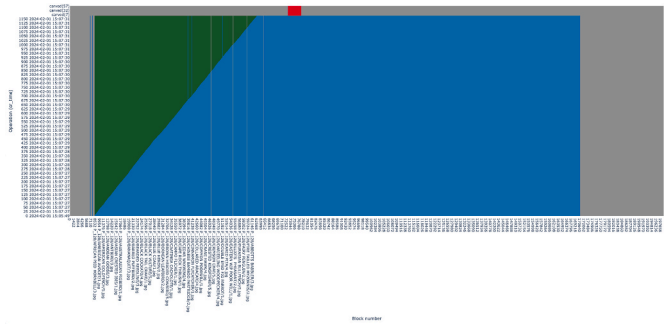


Fig. 11. Shows visualisation of FAT32 (scenario one) with carved data located outside FSUB (see top row).

incriminating data themselves within the current file system. Here a file was stored within a FAT file system but file operation circumstances mean that links to that file from within the file system have been overwritten. Fig. 12 shows the visualisation of this disk image, where the carved content is embedded within data from the current file system. Earlier experiments have shown that for FAT32, on Windows, the only old data that resides below the FSUB is in cluster slack of written files, meaning a maximum of 512 bytes for this example (1024 cluster size). Therefore, all data available suggests that this recovered content is likely to have been in the current file system. This is a significant development as a case can now be made that using these techniques, data recovered and situated in these circumstances could potentially be considered as evidence.

Furthermore, Fig. 13 shows the ability to zoom into the blocks containing the carved data of interest. Prior to those blocks, the creation of `/IVORY GULL/2.jpg` can be seen, and afterwards, the creation of `/IVORY GULL/4.jpg`. There are two hypotheses at this point. First, that

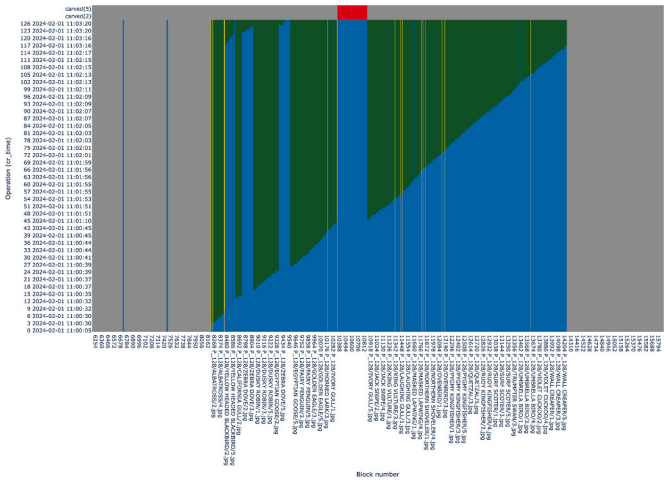


Fig. 12. Shows visualisation of FAT32 (scenario two) with carved data located inside FSUB (see top row).

the data of interest was created in between the times of 2.jpg and 4.jpg, or that a file was previously in those blocks at that time, was subsequently deleted, and then the carved file was written there at a later date, then deleted. In either case, some timing information can be determined that it was created after file 2.jpg. Looking at the raw data and noting the file is the exact size of the gap between 2.jpg and 4.jpg, requires that for hypothesis 2 the original file created was the same size (to the nearest block) of the recovered carved data. Reasoning about bounded timing information is weaker than the attribution to the current file system, but it is still a significant development in potential digital stratigraphy techniques.

8. Limitations and future work

In this paper, sixty experiments on FAT32, exFAT and NTFS using both Windows 11 and Debian 11 file system drivers have been conducted. A File System Activity Simulator was created and used to replicate actions that a user may carry out. While a large number of experiments was conducted, further permutations of creations, deletions and modifications are always possible to further test the conclusions drawn in this paper, with larger numbers of operations over a longer period of time. Furthermore, other versions of Windows, Linux and the file system drivers could be tested, as well as currently excluded platforms like macOS, or file systems such as APFS. Also, since the experiments presented in this paper have been conducted on virtual machines using virtual disks, real removable storage media could also be tested, although the hardware implementation should be abstracted away, and the small number of tests using *df_digger* on real USB devices showed similar results.

While the conclusions for FAT32 have been established to the point that a demonstration is possible, and a similar example could be constructed for exFAT, for NTFS, the complexity of the allocation algorithm suggests additional larger-scale simulations should be conducted to further understand the consequences of data in non-allocated space on NTFS on removable media. There are likely improvements to the FSUB concept with file systems such as NTFS that do not have a simple linear progression with cluster allocation and have metadata structures in the middle of the volume.

The FSUB calculation could also be further improved since, currently, the live files and deleted files with recoverable metadata are processed, and the highest allocation block is taken. As discussed earlier, it is also possible to add on file size to that, but fragmentation may introduce errors. Advanced techniques, such as using options presented in Section 1, such as identifying a separate carved file with internal metadata linkable to the suspect, may allow alternative methods of establishing the FSUB, allowing a carved file without metadata to be considered.

In terms of development practicalities, the use of *df_digger* on larger FAT32 volumes is currently a challenge due to the underlying *pytsk* technology, which works in sectors rather than clusters for FAT32 and exFAT file systems. Larger volumes therefore generate a number of data points that currently cannot be plotted. Options have been implemented to mitigate this, allowing max_blocks or max_files to be capped, which allows specific file system areas to be examined closely, but a better solution is needed. Furthermore, because of the usage of *pytsk*, currently, only TSK-supported file systems can be simulated and analysed with this framework.

Finally, despite the work in this paper, there may still be legal barriers or perceptions to overcome regarding the use of data from non-allocated space, even with additional work to strengthen the conclusions.

9. Conclusions

This paper set out to investigate if incriminating recovered content is identified in non-allocated space on removable storage media that

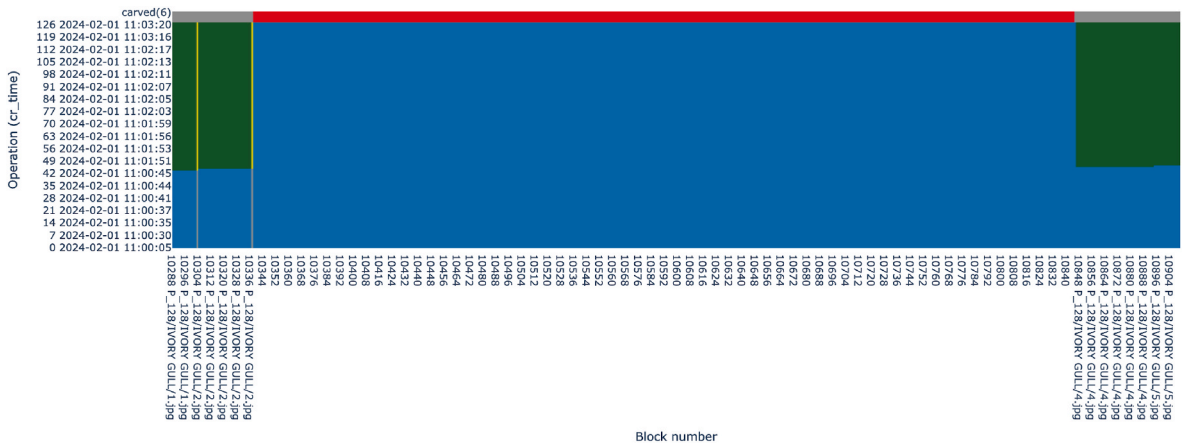


Fig. 13. Shows the identified carved content in context, with the files associated with the adjacent blocks shown (2.jpg and 4.jpg).

potentially contains old data, is it possible to differentiate whether that data was deleted from the current file system or potentially was already present.

To do this, the concept of a File System Upper Bound has been defined, and experiments conducted to verify that it can be calculated from a final disk image. It has been determined that it should represent a conservative estimate of how far into the storage media data has been written.

The research has also shown that residues of old data could be found prior to the FSUB for FAT32 and NTFS, but the amounts and consequences vary. For FAT32 on Windows, small amounts are found within cluster slack of files, but this has been shown to be somewhat mitigated for the purposes of this paper. For NTFS, it is more complicated as potentially large areas of the volume could contain old data, however, future work may be able to improve of the FSUB method and move to multiple masks for more complex file systems. It has also shown that there are significant differences between file system driver behaviour and that care is needed to understand the host OS when making inferences from file system artifacts.

However, experiments regarding content identified beyond the FSUB are mixed. In the experiments where the true FSUB is recorded at every stage, by definition, no data from the live file system was written beyond that. However, when a FSUB is calculated from a disk image, because files are deleted and metadata references to that content may be overwritten, at present, it is difficult to say that content beyond that FSUB was or was not part of the current file system.

However, in the demonstration section, examples have shown that despite previous work on recycled removable storage media and their components, suggesting that data in non-allocated space should be treated with caution, this work has shown a technique that, based on the experiments conducted, attribution of some data to the current file system could be achieved. It has even shown the potential to provide some approximate timing information to that previously metadata-free carved content.

Therefore, this could prevent important evidence from such media from being disregarded, and also shows the need for substantial further work into the application of stratigraphy concepts to digital forensics.

The code, datasets and additional information are available at https://github.com/janineschneider/Digital-Stratigraphy.

CRediT authorship contribution statement

Janine Schneider: Methodology, Software, Validation, Formal Analysis, Investigation, Resources, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization, Funding acquisition. **Maximilian Eichhorn:** Methodology, Software, Validation, Formal Analysis, Investigation, Resources, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Lisa Marie Dreier:** Writing - Original Draft, Writing - Review & Editing. **Christopher Hargreaves:** Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization, Supervision, Project administration.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work has been supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 “Cybercrime and Forensic Computing” (grant number 393 541 319/GRK2475/1-2019).

Appendix

Disk Formatting Behaviour (Section 3)

For this experiment, a virtual disk was initialised with the pattern ’this is old data’ written repeatedly from sector 0 to LBAmax. This baseline disk was then attached to a Windows 10 virtual machine and formatted with permutations of MBR, FAT32 and NTFS, using the ‘quick format’ option in the Windows formatting tool, both ticked and unticked.

As illustrated in Tables 14 and 15 for FAT32 (NTFS partitioning did not change the partition based results), the ‘quick format’ option writes the essential partition structures at the start of the disk, and the essential file system structures in the partition for the file system type that has been created. Outside of the partition created, using either option, old data remains between the MBR and the start of the first partition, and after the final partition. The key difference between the two options is within the partition/file system itself. Without ‘quick format’, a search for ‘old data’ produced no results and non-file system areas had been zeroed, whereas with the ‘quick format’ option, sectors containing ‘this is old data’ remained. From a formatting perspective, at least, this shows that using either formatting option, old data could be found when examining media, and it is much more likely if the ‘quick format’ option was used. To understand this further, it is necessary to consider the file system driver behaviour, which applies to data found within the current partition.

Table 14
Example MBR, FAT32 configuration with 'quick format' selected.

Start	End	Details
0	0	Boot sector + MBR
1	127	**Old data**
128	505983	Partition, contains core file system structures and **old data**
505984	511999	**Old data**

Table 15
Example MBR, FAT32 configuration with 'quick format' deselected.

Start	End	Details
0	0	Boot sector + MBR
1	127	**Old data**
128	505983	Partition, contains core file system structures and **zeroed data**
505984	511999	**Old data**

Description of selected experiments conducted (Section 5)

As part of the paper, various experiments have been conducted which are explained below in greater detail. The experiments were carried out for the FAT32, exFAT and NTFS file systems on the Debian 11 and Windows 11 operating systems. The operating systems were installed on virtual machines, which were assigned corresponding virtual hard disks on which the experiments were carried out. The virtual hard disks were dimensioned at 0.25 GB, as the experiments generate large output data, and the runtime increases drastically with larger file systems.

95p In the experiments labelled *95p*, only files in the file system are created in the individual operations. An 8-byte pattern consisting of the file ID is created as the file content of the created files. For the file with the id one and the file name *file_001.txt*, the pattern *f0000001* is selected as the file content and repeated according to the file size. The file sizes for the operations are selected randomly from the interval of 1 MB–3 MB. Files of this size are written until a usage of the file system of approx. 95 % is reached. Corresponding to a disk size of 0.25 GB, 117 files were created with the XML file we used.

95p_dir Like the 95p experiments, files are created in the operations with *95p_dir*. However, the operation *create directory* is added in those experiments. As the directories use less memory than the files created, the number of operations increases to 231 and 114 files are created. To do justice to realistic utilisation, the file operations are weighted higher than the directory operations, but the choice of operation is still random.

95p_del_50p In the *95p_del_50p* experiments, the deletion of files is now added as an operation for the first time, whereas the directory operations are omitted. These experiments are intended to depict a scenario in which files are constantly written to the storage medium until its usage reaches 95 %. From this point onwards, the most recently created files are deleted until a utilisation of 50 % is reached again. For example, this can occur if a camera's SD card is full and the user has to free up memory to take new pictures. In the XML file actually used, 118 files are initially created, 52 of which are deleted again.

80p_del In the experiments labelled *80p_del*, the file operations *create* and *delete* are selected at random, whereby a weighting is also selected here to create files more frequently than delete them. Furthermore, directory operations are not used. The resulting XML for the given parameters and a final file system utilisation of approx. 80 % comprised 207 operations, and a total of 152 files were created.

60p_dir_del The final experiments *60p_dir_del* include the file and directory operations *create* and *delete*. As directories can also be deleted in these experiments, it can happen that large memory areas are released for later use in an operation. This is because when a directory is marked as deleted, all subdirectories and files in it are also marked as deleted. To counteract the resulting drastic increase in the number of operations, a final file system usage of only 60 % is aimed for. For the specific XML file used, this results in 157 operations with a total of 105 files created.

Formatting Depending on the operating and file system, there are either two formatting options ('quick' and 'normal') or only one ('quick'). In cases where quick formatting is undesirable but unavoidable, the hard drive is zeroed out before formatting. If it should be examined whether old data can be observed after quick formatting, the hard drive is overwritten with a fixed pattern ('this is old data') before formatting. A regular expression can be used to search for the pattern.

Creation of evaluation scenario one (Section 7)

This scenario represents the situation where the files of interest were on a previous file system that has been subsequently formatted. A 100 MB FAT32 VHD was created in Windows 10. The following sequence of events was carried out: A dataset of bird images and a dataset of weather images was obtained from Kaggle.[3] A rhino picture was also obtained from Wikipedia, and the EXIF data was removed.

- The device was formatted with FAT32
- Pictures from the 'rain' subfolder were copied from the weather dataset
- The rhino image was copied to the root directory
- Pictures from the 'snow' subfolder were copied from the weather dataset
- the partition was formatted, again to FAT32 using the quick format option
- A set of folders and files were copied from the birds dataset

[3] https://www.kaggle.com/datasets/jehanbhathena/weather-dataset.

Creation of evaluation scenario two (Section 7)

This scenario represents the situation where the files of interest were located within the current file system, but deleted, and circumstances mean that connecting this deleted data to the current file system is difficult. A 100 MB MBR FAT32 VHD was created in Windows 10. The following sequence of events was applied: A dataset of bird images from Kaggle[4] was created, with selected files interspersed in the dataset from Wikipedia.

- Copied folders ALBERTROSS-HORNED LARK
- Created folder for IVORY GULL
- Copied files 1.jpg-5.jpg individually to IVORY GULL.
- Copied _tempfile1.txt - _tempfile25.txt (0bytes files) to fill up the directory entries.
- Copied folders JACK SNIPE - WALL CREAPER
- Deleted files (shift deleted so no recycle bin): BALD EAGLE/1–5.jpg, CALIFORNIA GULL/3.jpg, EGYPTION GOOSE/3.jpg, IVORY GULL/3.jpg
- Copied _temp26.txt to IVORY GULL top overwrite directory entry for 3.jpg
- Copied folders YELLOW HEADED BLACKBIRD, and ZEBRA DOVE

References

Bahjat, A., Jones, J., 2023. File allocation chronology and its impact on digital forensics. In: 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, pp. 612–618. https://doi.org/10.1109/CCWC57344.2023.10099265.

Bahjat, A.A., Jones, J., 2019. Deleted file fragment dating by analysis of allocated neighbors. Digit. Invest. 28, S60–S67. https://doi.org/10.1016/j.diin.2019.01.015. URL: https://www.sciencedirect.com/science/article/pii/S1742287619300258.

Bojic, N., Lambertz, M., Hilgert, J.N., 2022. Fsstratify: A Framework to Generate Used File Systems. Poster presented at dfrws eu. URL: https://dfrws.org/wp-content/uploads/2022/03/poster_Lambertz_20220307_1.pdf.

Casey, E., 2004. Section 9.6.4: digital stratigraphy. In: Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet. Academic press, pp. 247–249.

Casey, E., 2018. Digital stratigraphy: contextual analysis of file system traces in forensic. Science 63, 1383–1391. https://doi.org/10.1111/1556-4029.13722. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1556-4029.13722.

Casey, E., Nelson, A., Hyde, J., 2019. Standardization of file recovery classification and authentication. Digit. Invest. 31, 100873.

Freiling, F.C., Holz, T., Mink, M., 2008. Reconstructing people's lives: a case study in teaching forensic computing. In: Proceedings of the 4th International Conference on IT Incident Management & IT Forensics (IMF 2008). Gesellschaft für Informatik e.V.

Garfinkel, S., Shelat, A., 2003. Remembrance of Data Passed: A Study of Disk Sanitization Practices 1, pp. 17–27. https://doi.org/10.1109/MSECP.2003.1176992.

Hargreaves, C., Chivers, H., 2010. Detecting hidden encrypted volumes. In: Communications and Multimedia Security: 11th IFIP TC 6/TC 11 International Conference, CMS 2010, Linz, Austria, May 31–June 2, 2010. Proceedings 11. Springer, pp. 233–244.

Hargreaves, C.J., 2013. Visualisation of allocated and unallocated data blocks in digital forensics. In: EISMC, pp. 133–143.

Harris, E.C., 1987. Principles of Archaeological Stratigraphy. Academic Press.

Karresand, M., Axelsson, S., Dyrkolbotn, G.O., 2019. Using NTFS cluster allocation behavior to find the location of user data. Digit. Invest. 29, S51–S60. https://doi.org/10.1016/j.diin.2019.04.018. URL: https://www.sciencedirect.com/science/article/pii/S1742287619301690.

Karresand, M., Axelsson, S., Dyrkolbotn, G.O., 2020a. Disk cluster allocation behavior in windows and NTFS. Mobile Network. Appl. 25, 248–258. https://doi.org/10.1007/s11036-019-01441-1.

Karresand, M., Dyrkolbotn, G.O., Axelsson, S., 2020b. An Empirical study of the NTFS cluster allocation behavior over time. Forensic Sci. Int.: Digit. Invest. 33, 301008 https://doi.org/10.1016/j.fsidi.2020.301008. URL: https://www.sciencedirect.com/science/article/pii/S2666281720302572.

Karresand, N.M.M., 2023. Digital Forensic Usage of the Inherent Structures in NTFS. NTNU.

Lee, W.Y., Kwon, H., Lee, H., 2015. Comments on the Linux FAT32 allocator and file creation order reconstruction [Digit Investig 11(4), 224–233]. Digit. Invest. 15, 119–123. Special Issue: Big Data and Intelligent Data Analysis.

Li, Q., Zhang, Q., Tan, Y.a., Li, Y., Zheng, J., 2016. Research on allocator strategy of fat32 file system based on linux & windows. In: 2016 International Conference on Intelligent Control and Computer Application (ICCA 2016). Atlantis Press, pp. 248–252.

Lyell, C., 1872. Principles of Geology. Univ. of Chicago Pr.

Minnaard, W., 2014. The Linux FAT32 allocator and file creation order reconstruction. Digit. Invest. 11, 224–233. Special Issue: Embedded Forensics.

Moch, C., Freiling, F.C., 2009. The forensic image generator generator (forensig2). In: 2009 Fifth International Conference on IT Security Incident Management and IT Forensics. IEEE, pp. 78–93. https://doi.org/10.1109/IMF.2009.8.

Nordvik, R., Toolan, F., Axelsson, S., 2019. Using the object id index as an investigative approach for ntfs file systems. Digit. Invest. 28, S30–S39.

Pal, A., Memon, N., 2009. The evolution of file carving. IEEE Signal Process. Mag. 26, 59–71. https://doi.org/10.1109/MSP.2008.931081.

Patterson, J., Hargreaves, C., 2012. The Potential for Cross-Drive Analysis Using Automated Digital Forensic Timelines.

Schneider, J., Lautner, I., Moussa, D., Wolf, J., Scheler, N., Freiling, F., Haasnoot, J., Henseler, H., Malik, S., Morgenstern, H., Westman, M., 2021. In search of lost data: a study of flash sanitization practices. In: Proceedings of the Digital Forensics Research Conference Europe (DFRWS EU) 2021.

Tse, W.H.K., 2011. Forensic Analysis Using FAT32 File Cluster Allocation Patterns. Master's thesis. University of Hong Kong. URL: http://hdl.handle.net/10722/143258.

Vollebregt, Y., 2019. File Dating Based on the Physical Location of the File. Bachelor's Thesis. Open University of the Netherlands. URL: https://web.archive.org/web/20220526192803/ https://www.open.ou.nl/hjo/supervision/2019-y.vollebregt-bsc-thesis.pdf.

Willassen, S.Y., 2008. Finding evidence of antedating in digital investigations. In: 2008 Third International Conference on Availability, Reliability and Security. IEEE, pp. 26–32. https://doi.org/10.1109/ARES.2008.149. URL: https://ieeexplore.ieee.org/abstract/document/4529317.

[4] https://www.kaggle.com/datasets/gpiosenka/100-bird-species.