Mind the Slack? Reassessing the Relevance of File Slack in Modern Forensic Investigations

Anton Schwietert, Jan-Niclas Hilgert

Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE, Zanderstr. 5, 53177 Bonn, Germany

Abstract

File slack space has long been regarded as a potential source of residual data in digital forensic investigations. While prior work has established that slack exists in meaningful quantities on production systems and can be exploited for data hiding, the question of whether modern filesystems' own operations actually produce residual data in slack regions has not been systematically examined across today's diverse filesystem and operating system landscape. This paper presents a cross-platform empirical study of file slack space behavior across 12 filesystem implementations on Linux, macOS, and Windows. Through controlled detection and persistence experiments we examine whether residual data occurs in slack space and whether it survives common filesystem operations. Under these controlled conditions, the default filesystems of the three desktop platforms tested (NTFS on Windows, APFS on macOS, ext4 on Linux) cleared both RAM slack and drive slack across all tested file creation and modification scenarios. Beyond these defaults, behavior diverged substantially: FAT-based filesystems preserved drive slack, Copy-on-Write filesystems introduced "ghost slack" through block relocation, and the same filesystem specification yielded different slack behavior across operating systems and driver implementations. These findings demonstrate that slack space behavior is governed by the interplay of filesystem type, operating system, and driver implementation, and that its forensic value must be assessed on a per-configuration basis rather than assumed. We release our experimentation framework as open-source to support reproducibility.

Keywords:

1. Introduction

File slack space, the unused region between a file's logical end and the boundary of its last allocated cluster, has been regarded as a valuable forensic artifact since the earliest systematic treatments of digital evidence recovery Carrier (2005). Residual data in slack regions has historically been reported to reveal fragments of deleted files, expose hidden content, and provide traces of user activity that persist beyond standard file deletion Carrier (2005); Huebner et al. (2006). Forensic reference texts and commercial tools commonly include slack analysis as an investigative technique, and this practice has historically rested on the assumption that operating systems and their filesystem implementations do not systematically clear slack regions Carrier (2005); Huebner et al. (2006). For practitioners conducting triage on seized storage media, knowing whether slack analysis is likely to yield results on a given filesystem–OS combination directly affects evidence recovery strategy and resource allocation.

The forensic relevance of slack space can be decomposed into three distinct questions: *(a)* Does slack space exist in meaningful quantities on real systems? *(b)* Under what conditions does that slack space contain residual data? *(c)* Can slack be exploited to deliberately hide data? Previous work has provided affirmative answers to questions (a) and (c): Mulazzani et al. (2013) demonstrated that production Windows systems accumulate tens to hundreds of megabytes of slack, and numerous studies have shown that slack can serve as a covert storage channel Huebner et al. (2006); Liu et al. (2009); Srinivasan et al. (2024). Question (c) has been extensively investigated by prior work (as shown in Section 2) and is not revisited here. However, the empirical basis for question (b), whether the filesystem's own operations produce residual data in slack, rests on studies of legacy systems, predominantly FAT and early NTFS on Windows, and has not been systematically verified against the diverse filesystem and operating system landscape encountered in practice today. Although experienced practitioners might recognize that slack behavior is context-dependent, empirical evidence to guide this judgment across the modern filesystem landscape is lacking.

This paper addresses this gap through a cross-platform empirical study of file slack space behavior across 12 filesystem implementations on Linux, macOS, and Windows. Our contributions are:

1. A systematic empirical analysis of slack space detection and persistence across 12 filesystem–OS configurations, examining whether residual data occurs in

Email addresses: anton.schwietert@fkie.fraunhofer.de (Anton Schwietert), jan-niclas.hilgert@fkie.fraunhofer.de (Jan-Niclas Hilgert)

slack regions and whether it survives common filesystem operations.

2. A cross-platform comparison showing that slack behavior is determined by the filesystem–OS–driver combination rather than the filesystem type alone.

3. An open-source experimentation framework for reproducible file slack analysis across platforms and filesystems.

2. Background and Related Work

File systems allocate storage in fixed-size units, commonly called clusters or blocks Carrier (2005). When a file does not fill its last cluster, the remaining space is termed *file slack*. File slack is conventionally subdivided into *RAM slack* (the unused bytes between the file's logical end and the next sector boundary) and *drive slack* (the remaining bytes from that sector boundary to the cluster boundary) Carrier (2005). Slack may also arise in metadata structures Göbel and Baier (2018b); Schwietert and Hilgert (2025) and at volume or partition boundaries Berghel et al. (2008). This work focuses exclusively on file slack and uses the term *cluster* throughout for consistency.

2.1. Slack Space in File Systems

Within file systems, slack space manifests in different forms depending on whether it arises from file data allocation or from internal metadata structures.

File Slack. Srinivasan et al. (2024) present *slackFS*, a framework that hides entire filesystem volumes within the slack space of another filesystem. Their experiments quantify both the availability and stability of slack space in ext4 file systems. Their experiments show that an initial Ubuntu ext4 installation already contains several hundred megabytes of slack space. They further examine over 100,000 files under */usr* and show that they have remained unmodified for more than two years, highlighting this slack space as a persistent storage medium. The authors additionally demonstrate that, through erasure coding, hidden data can be reconstructed even if individual slack fragments are lost.

Metadata Slack. Beyond slack created at the end of a file, numerous studies have revealed slack space within internal file system structures. Research on ext-family file systems and XFS has demonstrated that unused bytes may appear in superblocks Piper et al. (2005); Göbel and Baier (2018a); Toolan and Humphries (2025b), block and inode bitmaps Göbel and Baier (2018a), group descriptor tables Piper et al. (2005); Göbel and Baier (2018a), and inode structures themselves Göbel and Baier (2018a); Toolan and Humphries (2025b). These works frequently incorporate empirical analyses demonstrating that such structural slack areas can contain traces of previous metadata or user data.

Recent studies have also shown that Copy-on-Write (CoW) file systems exhibit similar slack characteristics. For APFS, slack may occur in metadata regions such as container and volume superblocks or object maps Koolhaas and van Steenbergen (2020). Experiments on Btrfs have shown that, in addition to conventional file slack, underutilization inside internal nodes of its B-tree layout can lead to slack space Göbel et al. (2024). These observations indicate that slack is not an artifact of legacy storage designs but continues to manifest across contemporary file systems with sophisticated allocation and metadata management strategies.

2.2. Slack Handling

Different operating systems and file systems appear to handle slack space in varying ways. As previously mentioned, the literature commonly distinguishes file slack into *RAM slack* and *drive slack*. Across multiple sources, there is broad agreement that *RAM slack* is typically zeroed by native filesystem drivers in modern desktop operating systems, preventing the leakage of residual memory contents Mulazzani et al. (2013); Huebner et al. (2006); Larson (2009); Carlton (2005), though as our experiments will demonstrate, this generalization does not hold for all driver implementations. Carrier noted that, as of 2005, contemporary systems no longer copied arbitrary RAM data into the final sector of a file Carrier (2005).

In contrast, the described handling of *drive slack* remains inconsistent in the literature. Göbel and Baier report that certain Linux ext4 implementations explicitly zero portions of drive slack, reducing the leakage of residual data Göbel and Baier (2018b). Carrier also notes that some operating systems zero the *drive slack* region, although he does not specify which particular systems or filesystems exhibit this behavior Carrier (2005).

Conversely, several studies note that systems do not zero *drive slack*, leaving previous data recoverable under forensic examination Thampy et al. (2018); Larson (2009). Work on APFS further suggests that slack regions in Apple's filesystem may contain leftover metadata or user data unless explicitly sanitized Koolhaas and van Steenbergen (2020).

2.3. Slack Space from a Forensic Perspective

From a forensic standpoint, slack space represents both a vulnerability and a potential source of evidence. It can serve as a covert channel for hidden data as well as a residual storage area that may persist beyond standard file deletion or modification processes.

2.3.1. Data Hiding

Slack space has long been exploited for covert data storage. Early works by Huebner et al. (2006) and Liu et al. (2009) demonstrated file slack misuse in FAT and

NTFS, leading to modern frameworks like *slackFS* Srinivasan et al. (2024), which embeds entire hidden volumes within file slack. Beyond NTFS and FAT, subsequent studies extended slack-based hiding to ext-based systems Piper et al. (2005); Göbel and Baier (2018a), exFAT Heeger et al. (2021), XFS Toolan and Humphries (2025b), and APFS Koolhaas and van Steenbergen (2020), targeting both file slack and metadata structure slack. Göbel et al. (2024) showed that Btrfs generates slack in fixed 16 KiB nodes, while Schwietert and Hilgert (2025) and Toolan and Humphries (2025a) demonstrated intentional slack generation through non-standard sector sizes and symbolic link overflow.

These studies confirm that slack space offers a stealthy and technically feasible channel for data hiding, particularly in anti-forensic scenarios.

2.3.2. Recoverability of Slack Data

From a forensic recovery standpoint, slack space represents a valuable source of residual data. Garfinkel and Malan (2006) showed that discarded hard drives often contained recoverable slack data, including sensitive personal and corporate information. Several scientific studies have examined the phenomenon of slack space, analyzing its occurrence across a wide range of systems. Mulazzani et al. (2013) conducted the first systematic large-scale measurement study of file slack on Windows systems (NTFS, Windows XP–7). Their experiments showed that standard Windows installations using 4 KiB clusters accumulate tens to hundreds of megabytes of slack space, and that up to 78% of slack contents can remain intact even after major system updates.

Further evidence of the persistence of slack data is provided by Göbel and Baier (2018a), who assessed the detectability and durability of hidden content in ext4 slack space. Their findings indicate that even with routine file system use, remnants of data often remain accessible in file slack or directory entry slack, highlighting their forensic significance.

Koolhaas and van Steenbergen (2020) mention that slack data in APFS is highly volatile, as APFS' versioning mechanism quickly discards or zeroes out older metadata blocks whenever the filesystem is mounted and written to. Consequently, any data hidden in superblock, OMAP, or inode slack is short-lived and becomes unrecoverable once new checkpoints are created. This makes slack-based hiding techniques inherently unstable on live systems.

Collectively, these works establish slack space as a forensic asset, enabling partial or full recovery of previously deleted or hidden content, and revealing traces of user activity that might otherwise be considered unrecoverable.

3. Methodology

This section describes the systematic approach used to analyze slack space behavior across different filesystems and operating systems. The methodology consists of two complementary experiment types: detection experiments that identify residual data in slack space regions, and persistence experiments that evaluate whether such data survives various filesystem operations.

3.1. Detection Experiments

Detection experiments test whether residual data appears in file slack under various scenarios. Four distinct experiments were conducted to examine different scenarios in which data might persist in slack space regions. Table 1 provides an overview of these experiments.

All detection experiments follow a common methodology: files are created or manipulated according to the experiment scenario, after which complete snapshots of the file's allocated storage space are captured. These snapshots include both the logical file content and all slack space regions. RAM slack and drive slack are analyzed separately.

Each experiment classifies the slack space content into distinct categories. Slack space may contain identifiable residual data from previous operations, indicating data persistence. Alternatively, it may be zero-filled, suggesting the filesystem actively cleared the space. In addition, a distinction can be made between RAM slack and drive slack, as residual data may persist in one region while the other has already been cleared.

Experiment *D00* serves as a validation baseline, confirming that the tool can reliably detect and extract slack space artifacts when they are known to exist. Experiments *D01A* and *D01B* examine two forms of cluster reuse: *D01A* tests whether data from a previously deleted file leaks into the slack of a newly created file that reuses the same clusters, while *D01B* tests whether pre-existing data on a volume, written before the filesystem was created, persists in the slack of files created on the freshly formatted filesystem. Together, these two experiments cover the principal scenarios in which residual data could appear in file slack through cluster allocation. Experiment *D02* examines a distinct mechanism: file truncation, where reducing a file's size may expose previously written data in the newly created slack region (Table 1).

Testing CoW filesystems such as ZFS, APFS, and btrfs required careful experimental design to ensure that detection experiments measured actual slack space behavior rather than CoW relocation artifacts. In traditional filesystems, operations like file truncation or cluster reuse modify blocks in place; in CoW systems, these same operations may trigger block reallocation, fundamentally altering what the experiment measures. Dedicated experimental variants were therefore implemented to account for each filesystem's specific allocation behavior.

For cluster reuse experiments (*D01A*), we adapted the fill strategy to account for CoW allocation behavior. Standard filesystems typically reuse recently freed blocks from a free list, whereas CoW filesystems may prefer fresh blocks

Table 1: Detection Experiments

ID	Scenario	Research Question	Procedure
D00	Synthetic Slack Injection	Can the tool reliably detect artificially injected slack space data?	1. Create file with specific content 2. Inject known marker patterns into RAM slack 3. Inject known marker patterns into drive slack 4. Verify detection of injected patterns
D01A	Cluster Reuse After Deletion	Does deleted file data survive when clusters are reallocated?	1. Create source file with identifiable content 2. Delete source file 3. Store smaller target file in same location 4. Verificate block reusage 5. Check if deleted content appears in slack
D01B	Pre-filled Disk	Does pre-existing disk data appear in slack after filesystem creation?	1. Fill raw disk image with known pattern 2. Create fresh filesystem on disk 3. Create small file on new filesystem 4. Check if prefill pattern appears in slack
D02	Single Cluster Truncation	Does file data survive in slack after truncating within one cluster?	1. Create file smaller than one cluster 2. Truncate to smaller size within same cluster 3. Check if truncated data remains in slack

or allocate from different regions to maintain snapshot consistency. To force actual cluster reuse in btrfs, ZFS, and APFS, we employed a fragmented fill strategy: creating many small files (100 KB each) and deleting every other one to create allocation "holes" that subsequent writes must fill. This ensured that new file allocations genuinely reused previously occupied blocks rather than simply allocating from unused disk regions.

For truncation experiments, we verified that the file continued to reference the same physical block after truncation by comparing block addresses before and after the operation. This verification was critical because CoW filesystems might relocate blocks during metadata updates, transforming what should be an in-place slack preservation test into a block relocation scenario.

ZFS required experimental adaptations due to its variable block allocation strategy based on the *recordsize* parameter (configured as 4 KB). Files $\leq$ *recordsize* are allocated as single blocks rounded to 512-byte sector boundaries, producing RAM slack but no drive slack. Files $>$ *recordsize* span multiple full *recordsize* blocks, with the final block padded to *recordsize*, producing both RAM and drive slack.

Standard detection experiments used 3000-byte files ($<$ 4096-byte *recordsize*). To test drive slack behavior, we created large file variants (*d01a_zfs_large*, *d01b_zfs_large*) using 5096-byte files, forcing two-block allocations where the final block contained both RAM slack and drive slack. This distinction is forensically significant: small ZFS files offer minimal slack space (RAM only), while large files may contain substantial drive slack regions.

3.2. Persistence Experiments

Persistence experiments test whether slack space data survives filesystem operations. These experiments receive a disk image from a successful detection run, where slack space is known to contain residual data. Each persistence experiment operates on an independent copy to prevent interference. Table 2 provides an overview of all persistence experiments.

The experimental procedure follows a consistent pattern. First, the pre-operation state of the slack space regions is recorded. Next, the specified filesystem operation is performed. Finally, the same regions are analyzed to determine whether the data persisted, was partially cleared, or was completely removed.

For CoW filesystems, an additional step identifies whether storage blocks were relocated instead of being modified in place. This distinction is essential, as block relocation fundamentally alters the interpretation of data persistence.

Experiment *P04* tests a pure metadata operation. Experiments *P05A-B* address forensically relevant scenarios where system maintenance tools may inadvertently modify or clear slack space during routine operations. Filesystem check experiments utilized platform-specific consistency checking tools: *fsck* variants (*fsck.ext4*, *fsck.vfat*, *fsck.exfat*, *xfs_repair*, *btrfs_check*) on Linux, *fsck_apfs*, *fsck_hfs*, and *fsck_msdos* on macOS, and *chkdsk* on Windows. Each tool was executed in both read-only verification mode and repair mode where supported by the respective tool.

3.3. Experimental Environment

The experiments were conducted on Kali GNU/Linux Rolling (kernel 6.16.8), macOS Sequoia 15.2, and Windows 11 Pro. Testing covered diverse filesystem implementations to capture varying slack space behaviors. Table 3 provides a detailed overview of the tested filesystem–operating system combinations.

For NTFS testing, ntfs-3g was used on Linux to evaluate this Windows filesystem in a cross-platform context. *FUSE*-based filesystems were included to assess whether user-space filesystem implementations exhibit different slack space behavior compared to native kernel-level drivers.

Table 2: Persistence Experiments

ID	Operation	Research Question	Procedure
P01A	Append 1 Byte	Does slack survive appending minimal data?	Append single byte to file end
P01B	Append to Sector	Does drive slack survive filling RAM slack completely?	Append enough bytes to reach sector boundary
P01C	Append into Drive	Does slack survive when append crosses into drive slack?	Append 512+ bytes beyond sector boundary
P02A	Overwrite First Byte	Does modifying file start affect slack?	Overwrite first byte of file
P02B	Overwrite Last Byte	Does modifying file end affect slack?	Overwrite last byte of file content
P03A	Copy with cp	Does standard copying preserve or clear slack?	Copy file using cp command
P03B	Copy no reflink	Does forced non-reflink copy affect slack differently?	Copy file using cp –reflink=never
P04	Rename	Does metadata-only operation affect slack?	Rename file within same directory
P05A	Filesystem Check	Does read-only check affect slack?	Run fsck in check-only mode
P05B	Filesystem Repair	Does repair mode modify slack?	Run fsck in repair mode

FUSE filesystems, exclusively supported on Linux, run in user space and communicate with the kernel through a well-defined API Vangoor et al. (2017).

All experiments were conducted on disk images, isolating filesystem behavior from storage-layer effects. On flash-based media, TRIM may zero freed blocks before the filesystem reuses them, potentially eliminating residual data that would otherwise appear in slack upon reallocation (experiments *D01A-B*). Experiment *D02* is unaffected, as the cluster remains allocated throughout.

Filesystem	Linux	macOS	Windows
APFS	✗	✓	✗
btrfs	✓	✗	✗
exFAT	✓ †	✓	✓
ext2/3/4	✓	✗	✗
FAT	✓ †	✓	✓
HFS+	✗	✓	✗
NTFS	✓ †	✗	✓
XFS	✓	✗	✗
ZFS	✓	✗	✗

† Additionally tested via FUSE on Linux.

Table 3: Tested filesystem–OS combinations.

3.4. Data Collection and Analysis

Each experiment generates comprehensive artifacts for verification. Hexdumps capture exact byte content before and after operations. *Json* metadata files document pattern match analysis with precise percentages. Block extraction data records physical disk locations.

Pattern analysis employs byte-level comparison between expected patterns and actual slack content. The framework extracts byte ranges for RAM slack and drive slack regions, counts pattern matches, calculates percentages, and identifies boundaries where transitions occur.

Physical offset tracking uses filesystem-specific tools to map logical file blocks to physical disk locations. The *Sleuth Kit* handles ext2/3/4, NTFS, and FAT filesystems. Python *dissect* handles btrfs and xfs with their extent-based schemes. For ZFS, the native *zdb* tool provides physical locations. This multi-backend approach ensures accurate offset tracking across diverse architectures, which is essential for detecting Copy-on-Write behavior. The entire framework[1] which was used to automatically conduct all experiments is part of the contribution of this paper.

4. Results

This section presents the comprehensive findings of slack space behavior analysis across filesystems and operating systems. The results are organized into five sections that progressively examine slack space characteristics from standard configurations to specialized implementations and their practical implications.

We begin with Standard Filesystems, analyzing the default filesystem of each platform to establish baseline slack space behavior in typical computing environments. Then we examine whether alternative native filesystems on the same operating system exhibit different behavior. The following Cross-Platform Analysis examines filesystems available across multiple operating systems, revealing how the same filesystem specification exhibits different slack handling under different OS implementations. The Native vs. *FUSE* Filesystems section compares kernel-level and userspace filesystem drivers on Linux, identifying implementation-specific divergences in slack space management. Finally, Forensic Implications synthesizes these findings to assess the practical value and reliability of slack space analysis in modern digital forensics investigations.

4.1. Standard File Systems

To evaluate slack space behavior across different operating systems, we focused first on the standard filesystem of each platform: NTFS on Windows, APFS on macOS, and ext4 on Linux. Table A.8 presents a comprehensive overview of the detection experiment outcomes across these three filesystems.

[1] `https://anonymous.4open.science/r/mind-the-slack/`

4.1.1. Slack Detection

Across NTFS, APFS, and ext4, both cluster reuse experiments cleared all slack data. In *D01A*, clusters freed by file deletion were fully zeroed upon reallocation. In *D01B*, formatting alone left a substantial portion of prior data physically intact. However, the subsequent file creation operation completely zeroed the allocated clusters, eliminating all residual data from the slack regions.

The file truncation experiment (*D02*) analyzed slack space behavior during file size reduction. In all three filesystems, slack space was cleared in the tested truncation scenario, indicating that these filesystems apply clearing operations that encompass the slack region when file sizes are reduced within a single cluster.

Notably, although APFS is architecturally a CoW filesystem, our experiments show that it clears slack through in-place block zeroing rather than CoW relocation. Provenance analysis consistently reported the same physical block, making APFS indistinguishable from the other filesystems with respect to slack space behavior.

4.1.2. Slack Persistence

All detection experiments (*D01A–D02*) produced fully zeroed slack spaces across NTFS, ext4, and APFS. As a result, no naturally occurring residual data exists in these slack regions to evaluate persistence. To evaluate how slack data behaves under subsequent filesystem operations, we therefore rely on experiment *D00*, which synthetically injects known marker patterns directly into the RAM and drive slack regions of a file. This synthetic injection serves as the baseline for all persistence experiments (*P01–P05*): once slack data has been placed artificially, each persistence operation is applied and the slack region is re-examined. This approach explains why *D00* does not appear in the detection results (Table A.8). *D00* is not a detection experiment but a setup step that establishes the initial condition for persistence testing. As summarized in Table 4, all operations involving data rewriting, including append operations of varying sizes, overwrite operations at both the beginning and end of files, and file copying, consistently resulted in the complete removal of slack space data across all three standard filesystems.

In contrast, experiments *P04* and *P05* preserved slack space data. File renaming operations maintained slack content across all tested platforms, as these actions modify only directory entries without accessing the associated data blocks. Likewise, filesystem check and repair procedures preserved slack data, indicating that consistency checking utilities focus on metadata structures and allocation integrity rather than examining slack space regions.

In summary, under the tested scenarios, residual data did not appear in file slack on the default filesystems of Windows 11 (NTFS), macOS Sequoia (APFS), and Linux 6.16 (ext4). Neither form of cluster reuse nor file truncation produced recoverable residual data in slack space. Even when slack data was synthetically injected for persistence testing, the persistence proved to be extremely limited. All tested operations that modify file contents cleared slack space, resulting in a very short survival duration for any slack data.

Operation	Windows ntfs	macOS APFS	Linux ext4
Append 1 Byte	**Clr**	**Clr**	**Clr**
Append to Sector	**Clr**	**Clr**	**Clr**
Append > Sector	**Clr**	**Clr**	**Clr**
Overwrite Start	**Clr**	**Clr**	**Clr**
Overwrite End	**Clr**	**Clr**	**Clr**
Copy (cp)	**Clr**	**Clr**	**Clr**
Rename	**Surv**	**Surv**	**Surv**
fsck repair	**Surv**	**Surv**	**Surv**

Legend: Surv = Slack data survives, Clr = Slack data cleared.

Table 4: Slack space persistence across operating systems and standard file systems.

4.2. Intra-Platform Filesystem Comparison

Having established that the default filesystem of each platform consistently clears slack space across all detection experiments, we now examine whether alternative native filesystems on the same operating system exhibit different behavior. This comparison isolates the filesystem as the variable while holding the operating system and hardware constant. Table 5 summarizes the detection experiment outcomes.

Filesystem	Cluster Reuse				D02	
	D01A File Deletion		D01B Pre-existing Data		Truncate File	
Linux						
ext4 (default)	✗		✗		✗	
ext2/3	✗		✗		✗	
xfs	✗		✗		✗	
btrfs	✗		✗		✓	
zfs	✗		✗		✗	
Windows						
ntfs (default)	✗		✗		✗	
fat32	**R**	**D**	**R**	**D**	**R**	**D**
exfat	✗		**R**	**D**	**R**	**D**
macOS						
APFS (default)	✗		✗		✗	
HFS+	✗		✗		✓	

Legend: ✓ residual data in slack; ✗ slack cleared/zeroed; ***R*/*D*** RAM slack cleared, drive slack preserved.

Table 5: Detection experiment outcomes comparing default and alternative native filesystems within each platform.

4.2.1. Linux: ext Family and XFS

On Linux, the ext family (ext2/3, ext4) and XFS form a functionally uniform group with respect to slack space handling. All four filesystems consistently clear both RAM slack and drive slack across all detection experiments. This

uniformity persists despite significant architectural differences: ext2 lacks journaling, ext3 introduces journal-based crash recovery, ext4 adds extents and delayed allocation, and XFS employs a B-tree-based design with extent-based allocation. The consistent clearing behavior across architecturally diverse filesystems raises the possibility that zero-initialization may be applied at the Linux kernel's VFS or block I/O layer rather than by each filesystem individually, though confirming this hypothesis would require kernel source-code analysis.

Persistence experiments confirmed equally uniform behavior: all data-modifying operations (appending, overwriting, copying) clear slack space, while metadata-only operations (renaming, filesystem checks) preserve it. This matches the ext4 baseline described in the previous section.

Btrfs and ZFS, also natively available on Linux, diverge from this uniform pattern due to their Copy-on-Write architecture. Their behavior is detailed in the following subsection.

4.2.2. Linux: Copy-on-Write Filesystems (btrfs and ZFS)

Btrfs and ZFS share the same fundamental Copy-on-Write design principle, namely that data is never overwritten in place, yet they produce different experiment outcomes. In detection experiments, btrfs clears slack during both cluster reuse experiments, but preserves slack during file truncation (Table 5). This occurs because btrfs treats truncation as a metadata-only operation: reducing the file's logical extent without rewriting or reallocating the underlying block. ZFS, by contrast, clears slack data across all three detection experiments, including truncation. However, the mechanism differs fundamentally from in-place zeroing: ZFS relocates the block to a new physical offset during truncation, writing zeroed slack into the new copy while the original block remains unreferenced at its prior location. The provenance data confirms this distinction, recording different physical offsets before and after the operation.

The practical forensic consequence is that btrfs is the only Linux filesystem that preserves slack through truncation. An investigator examining a btrfs volume can expect through truncation generated slack data to remain intact and recoverable at the file's current block offset.

Persistence experiments reveal a second CoW-specific phenomenon: block relocation on data-modifying operations. On both btrfs and ZFS, appending or overwriting even a single byte triggers CoW, causing the entire block to be written to a new physical location. While the "current" block at the new offset contains zeroed slack, the "old" block at the original offset persists as unreferenced data. We define this as "ghost slack" that retains the prior slack content until the space is reclaimed. Metadata-only operations (renaming, filesystem checks) do not trigger CoW and slack data survives in place, consistent with non-CoW filesystems.

Copy operations introduce an additional CoW-specific behavior: on btrfs, the default `cp` command creates a *reflink* (a CoW clone sharing the same physical blocks), this leads to no useful experiment since source and destination reference identical physical data. When reflinking is explicitly disabled (`cp -reflink=never`), the destination receives a fresh block with cleared slack, while the source remains unmodified. ZFS exhibits the same *reflink* behavior for its default copy operation.

4.2.3. Windows: NTFS vs. FAT Family

The most pronounced intra-platform divergence emerges on Windows, where NTFS and the FAT-family filesystems exhibit fundamentally different write granularities. In our experiments, NTFS exhibited cluster-granular write behavior: modifications to a file resulted in the entire 4096-byte cluster being rewritten, zeroing all slack space in the process. FAT32 operates at sector granularity (512 bytes), updating only the sectors directly affected by a write operation. Consequently, drive slack remains intact through both forms of cluster reuse and file truncation alike.

This produces a characteristic forensic signature: the *R/D* pattern, where RAM slack is cleared while drive slack preserves residual data. FAT32 exhibits this pattern consistently across all detection experiments, including both cluster reuse experiments (*D01A-B*), making it the most consistently forensically permissive filesystem among the native implementations tested in this study. ExFAT occupies a middle ground: during cluster reuse after deletion (*D01A*), exFAT clears both RAM and drive slack like NTFS, but during file truncation (*D02*) and cluster reuse across filesystem creation (*D01B*), it reverts to the FAT32 sector-granular *R/D* pattern. This hybrid behavior suggests that exFAT implements allocation-time clearing comparable to NTFS but lacks truncation-time and format-time clearing.

Persistence experiments reinforce this distinction. On FAT32, even operations such as overwriting the first or last byte of a file clear only the affected sector's RAM slack, leaving drive slack fully intact. On NTFS, the same operations zero the entire cluster. Only metadata operations (renaming) and filesystem checks (*chkdsk*) preserve slack across all three Windows filesystems.

4.2.4. macOS: APFS vs. HFS+

On macOS, the comparison between APFS and its predecessor HFS+ reveals a single but forensically significant divergence. Both filesystems clear slack during both cluster reuse experiments. However, file truncation (*D02*) produces contrasting results: APFS clears all slack space, while HFS+ preserves both RAM and drive slack.

HFS+ treats file truncation as a pure metadata operation, adjusting the file's logical size without modifying the underlying data blocks. The physical content of the truncated region remains fully intact and recoverable. APFS, in contrast, actively zeros the slack region created by truncation. Persistence experiments confirm this pattern: on

HFS+, through truncation generated slack data survives all subsequent metadata operations and filesystem checks, persisting until a data-modifying write occurs. On APFS, no truncation-generated slack exists to persist.

Notably, despite being architecturally a CoW filesystem, APFS does not exhibit the block relocation behavior observed on btrfs and ZFS. As noted in the baseline analysis of standard filesystems, APFS consistently zeros slack in place at the same physical block offset.

4.3. Cross-Platform Analysis

To evaluate the influence of operating system implementation on slack space behavior, we conducted comparative experiments across Windows, macOS, and Linux using filesystems supported on multiple platforms. Table 6 presents the results for experiments *D01A*, *D01B*, and *D02*. *D00* is omitted because every filesystem and operating system combination universally preserves injected slack data. The per-OS baselines for each filesystem were established in Section 4.2; the discussion below focuses exclusively on cross-platform divergences.

OS	Cluster Reuse					
	D01A File Deletion		**D01B** Pre-existing Data		**D02** Truncate File	
exFAT						
Linux	**R**	**D**	**R**	**D**	**R**	**D**
macOS	✗		✗		✓	
Windows	✗		**R**	**D**	**R**	**D**
FAT32						
Linux	**R**	**D**	**R**	**D**	**R**	**D**
macOS	✗		✗		✓	
Windows	**R**	**D**	**R**	**D**	**R**	**D**
vfat						
Linux	**R**	**D**	**R**	**D**	**R**	**D**
macOS	✗		✗		✓	
NTFS						
Linux	✗		✗		✓	
Windows	✗		✗		✗	

Legend: ✓ residual data in slack; ✗ slack cleared/zeroed; ***R/D*** RAM slack is cleared but drive slack not.

Table 6: Cross-OS detection experiment outcomes. D00 (slack injection) is omitted as all combinations universally yield ✓.

4.3.1. FAT32 and vfat

On Linux and Windows, FAT32 and vfat behave identically across all three experiments: the characteristic *R/D* pattern (RAM slack cleared and drive slack not) appears consistently in *D01A-B* and *D02*. MacOS departs fundamentally from this pattern. During both cluster reuse experiments, macOS clears both RAM and drive slack. During truncation (*D02*), however, macOS preserves the full slack region. This opposing pattern is unique to macOS among the tested platforms. Notably, in all tested scenarios, macOS did not distinguish between RAM and drive slack on FAT filesystems, suggesting that its FAT driver operates at cluster granularity rather than sector granularity.

4.3.2. exFAT

ExFAT exhibits three distinct behavioral profiles across three operating systems. Linux applies the *R/D* pattern uniformly in *D01A-B* and *D02*, mirroring its FAT32 behavior. Windows diverges: *D01A* results in complete clearing, while *D01B* and *D02* revert to the *R/D* pattern. This indicates that the Windows exFAT driver employs more aggressive zeroing during cluster reallocation after deletion than during truncation or cluster reuse across filesystem creation. MacOS follows yet another pattern: complete clearing for both *D01A* and *D01B*, but full preservation for *D02*. The result is that the same filesystem specification yields measurably different slack behavior on each platform, confirming that the OS driver implementation affects whether residual data survives.

4.3.3. NTFS

NTFS was tested on Linux and Windows only, as macOS provides no native NTFS write support. Cluster reuse (*D01A-B*) produces consistent results: both platforms clear the entire slack region. The key divergence appears in truncation (*D02*): Linux preserves all slack data, while Windows clears it. On Linux, the ntfs-3g driver treats truncation as a metadata-only operation that updates the file size without touching the underlying data blocks. The Windows native NTFS driver, by contrast, zeros the slack region during truncation.

The cross-platform analysis demonstrates that slack space behavior is shaped by a three-dimensional interaction of filesystem specification, operating system, and driver implementation. The same FAT32 specification produces sector-granular clearing on Linux and Windows but whole-cluster clearing or preservation on macOS. exFAT yields three measurably different profiles across three OSes. The NTFS truncation divergence between ntfs-3g and the native Windows driver further illustrates that even ostensibly compatible implementations of the same filesystem can exhibit fundamentally different data-retention characteristics.

4.4. Native vs. FUSE

To assess whether the *FUSE* userspace driver layer affects slack handling, we compared native and *FUSE* implementations of NTFS, vfat, and exFAT on Linux (Table 7).

Table 7 presents the detection experiment outcomes comparing native and *FUSE* implementations of ntfs, vfat, and exfat filesystems under Linux.

4.4.1. NTFS

Both native ntfs and ntfs-3g *FUSE* implementations demonstrated identical slack space behavior across all detection experiments. As already observed in the previous

Filesystem	Cluster Reuse					
	D01A File Deletion		**D01B** Pre-existing Data		**D02** Truncate File	
ntfs	✗		✗		✓	
ntfs fuse	✗		✗		✓	
vfat	**R**	**D**	**R**	**D**	**R**	**D**
vfat fuse	**R**	**D**	**R**	**D**	✓	
exfat	**R**	**D**	**R**	**D**	**R**	**D**
exfat fuse	✓		✓		✓	

Legend: ✓ residual data in slack; ✗ slack cleared/zeroed; ***R/D*** RAM slack is cleared but drive slack not.

Table 7: Detection experiments outcomes in native and *FUSE* filesystems.

section, ntfs-3g distinguishes between experiment *D01A/B*, in which slack space is completely cleared, and experiment *D02*, in which slack space is fully preserved. This behavior is likewise adopted by ntfs *FUSE*.

Technical analysis of the ntfs-3g source code[2] reveals the mechanism behind this clearing behavior. When writing at the end of a file's initialized size, ntfs-3g intentionally zero-fills data to the cluster boundary. Code comments within the implementation explain this design choice as a performance optimization: by proactively zeroing to cluster boundaries, the driver prevents the kernel from having to seek and read existing disk blocks during write operations, thereby reducing I/O overhead.

4.4.2. vfat

In contrast to ntfs's implementation consistency, the FAT-family filesystems exhibited significant behavioral differences between native and *FUSE* variants, particularly in the exfat implementation. Both vfat implementations demonstrated the characteristic *R/D* pattern during experiments *D01A/B*, where RAM slack was cleared while drive slack remained preserved. However, the truncation experiment revealed a critical difference: while native vfat maintained the *R/D* pattern during file size reduction, vfat_fuse completely preserved all slack space data. This suggests that the *FUSE* implementation treats truncation more conservatively, avoiding any data clearing operations during metadata-only file modifications.

4.4.3. exfat

The exfat filesystem demonstrated the most implementation differences. Native exfat exhibited the standard FAT-family *R/D* pattern during both cluster reuse and file truncation operations, consistent with sector-granular clearing behavior. In contrast, exfat_fuse preserved all slack space data in both scenarios, showing complete data preservation across all detection experiments following the initial injection. This complete absence of clearing mechanisms in exfat_fuse, during both allocation and modification operations, represents an architectural difference from the native implementation. The exfat_fuse behavior suggests either an incomplete implementation of data clearing mechanisms or a deliberate design choice prioritizing functional compatibility over security-oriented data erasure.

A consistent pattern emerged across both fat *FUSE* implementations during the persistence experiments: slack data remained unchanged through subsequent file operations that would typically trigger clearing in native implementations. This behavior was observed in exfat_fuse, where even cluster reuse operations that cleared slack in native exfat left slack data intact. This preservation of slack data suggests that *FUSE* implementations may lack the low-level block management operations that native filesystems employ for comprehensive slack clearing.

One operation demonstrated complete consistency across all filesystem implementations, both native and *FUSE*: file copying via `cp` consistently cleared all slack space data in every tested configuration. This behavior reflects the fundamental nature of copy operations, which read file content up to the logical file size and write it to a new allocation, inherently excluding slack regions that exist beyond the file's official length. Among all tested operating system and filesystem combinations, no configuration exhibited a transfer of slack data during file copy operations.

4.5. Practical Implications

The following forensic implications summarize the practical consequences of these findings for digital investigations. To support informed forensic decision-making, the most relevant implications are outlined below.

The tested default filesystems (NTFS, APFS, ext4) did not produce residual data in file slack regions under the tested conditions. The default filesystems on Windows 11 (NTFS), macOS Sequoia (APFS), and Linux 6.16 (ext4) did not exhibit residual data in file slack areas under any of the controlled scenarios tested in this study. Under the conditions tested, slack data observed in these filesystems originated from artificial injection rather than the filesystem operations examined.

Slack behavior is highly inconsistent. Beyond standard filesystems, predicting the occurrence and persistence of slack space is inherently difficult. Slack handling varies substantially across different filesystems, operating systems, and even between different implementations of the same filesystem. Consequently, slack analysis must be evaluated on a per-filesystem and per-OS basis rather than relying on general assumptions.

FAT-based filesystems remain a viable slack artifact source on Windows and Linux. FAT-derived filesystems continue to produce forensically relevant artifacts in file slack regions, particularly within drive slack. While RAM slack is frequently cleared in native implementations, drive slack often persists and may contain recoverable data.

Copy-on-Write filesystems may produce "ghost slack." On btrfs and ZFS, data-modifying operations such

[2] `https://github.com/tuxera/ntfs-3g`

as appending or overwriting may trigger block relocation due to Copy-on-Write semantics. The original block, including its slack content, persists as unreferenced data at its previous physical location. These artifacts are not accessible through live filesystem analysis and require raw disk imaging and analysis of unallocated space for recovery.

Truncation is a forensic wildcard. The truncation experiments (*D02*) produced the most divergent results across filesystems. Truncation preserves slack on btrfs, HFS+, NTFS-on-Linux, and FAT-on-macOS, yet clears it on ext4, APFS, ZFS, and NTFS-on-Windows.

Standard file copy operations eliminated slack space artifacts across all tested configurations. Across all tested filesystems and operating systems, standard file copy operations consistently destroyed all slack data. In all tested cases, destination files were written using fresh allocations, and no residual slack content was observed in the destination files.

These findings provide empirical grounds for reassessing the role of slack space analysis in digital forensics. Slack space examination should not be treated as a broadly applicable investigative technique with uniformly high evidentiary yield. Instead, forensic practitioners should recognize slack space analysis as a highly conditional technique. Its effectiveness depends critically on the filesystem type, the operating system, the driver implementation (native vs. *FUSE*), and the specific sequence of operations performed on the system.

Consequently, the common assumption in forensic literature and training materials that file slack routinely contains recoverable residual data does not hold for the default filesystem configurations of current desktop platforms. For investigations involving these default configurations, the expected evidentiary yield of slack analysis is low, and practitioners should focus slack examination on the specific configurations identified in this study as producing recoverable artifacts, such as FAT-based removable media and post-truncation scenarios on susceptible filesystems.

5. Limitations and Future Work

The main methodological limitations of this study are as follows:

- **Allocation Unit Variations:** All experiments used a fixed cluster size of 4096 bytes and a sector size of 512 bytes.
- **Longitudinal OS Analysis:** Testing was limited to single OS versions.
- **Filesystem Coverage:** Encrypted storage configurations and filesystems with compression or deduplication enabled remain untested.

These limitations suggest several directions for future research. Varying allocation parameters, particularly examining 4 KiB physical sectors common on modern SSDs, would clarify whether the observed clearing patterns reflect fundamental filesystem design choices or are artifacts of a specific allocation geometry. Longitudinal studies across successive OS releases would reveal whether slack handling behavior is stable or evolves over time, and whether Linux distribution-specific kernel configurations introduce further variation. Extending the analysis to encrypted, compressed, or deduplicated storage configurations would clarify how additional storage abstraction layers affect slack persistence.

6. Conclusion

This work presented a systematic, cross-platform empirical study of file slack space behavior across 12 filesystem implementations on Linux, macOS, and Windows. Through controlled detection and persistence experiments, we examined whether residual data occurs in slack regions and how it behaves under common filesystem operations.

Under the tested conditions, the default filesystems of all three platforms, namely NTFS on Windows 11, APFS on macOS Sequoia, and ext4 on Linux 6.16, cleared both RAM slack and drive slack during file creation, cluster reuse, and truncation, producing no naturally occurring residual data in slack regions. Even synthetically injected slack data was eliminated by every tested content-modifying operation.

Beyond these defaults, behavior diverged substantially. FAT-based filesystems on Windows and Linux preserved drive slack across all tested scenarios, remaining the most forensically permissive family in our experiments. Copy-on-Write filesystems (btrfs, ZFS) introduced a distinct "ghost slack" phenomenon, in which block relocation leaves prior slack content at unreferenced physical offsets, recoverable only through raw disk imaging. Truncation produced the most inconsistent results across configurations, preserving slack on btrfs, HFS+, and NTFS-on-Linux while clearing it on ext4, APFS, ZFS, and NTFS-on-Windows. Critically, the same filesystem specification yielded measurably different slack behavior depending on the operating system and driver implementation, as demonstrated by exFAT producing three distinct behavioral profiles across three platforms and by divergences between native and *FUSE* drivers on Linux.

These results demonstrate that slack space behavior is governed by the interplay of filesystem type, operating system, and driver implementation rather than by any single factor. The common assumption that file slack routinely contains recoverable residual data is not supported by our experiments on current default configurations, though it remains applicable to FAT-based removable media and specific non-default scenarios. Forensic practitioners should treat slack analysis as a conditional technique whose expected yield depends on the specific filesystem–OS–driver combination under investigation. To support reproducibility and further research, we release the complete experimentation framework as open-source software.

References

Berghel, H., Hoelzer, D., Sthultz, M., 2008. Data hiding tactics for windows and unix file systems. Advances in Computers 74, 1–17.

Carlton, G.H., 2005. A critical evaluation of the treatment of deleted files in microsoft windows operating systems, in: Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05), IEEE. pp. 1–8. doi:10.1109/HICSS.2005.310.

Carrier, B., 2005. File system forensic analysis. Addison-Wesley Professional.

Garfinkel, S.L., Malan, D.J., 2006. One big file is not enough: A critical evaluation of the dominant free-space sanitization technique, in: Danezis, G., Golle, P. (Eds.), Privacy Enhancing Technologies. Springer, Berlin, Heidelberg. volume 4258 of *Lecture Notes in Computer Science*, pp. 135–151. doi:10.1007/11957454_8.

Göbel, T., Baier, H., 2018a. Anti-forensic capacity and detection rating of hidden data in the ext4 filesystem, in: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics XIV, Springer International Publishing, Cham. pp. 87–110.

Göbel, T., Baier, H., 2018b. Fishy-a framework for implementing filesystem-based data hiding techniques, in: International Conference on Digital Forensics and Cyber Crime, Springer. pp. 23–42.

Göbel, T., Baier, H., Türr, J., 2024. Generating usable and assessable datasets containing anti-forensic traces at the filesystem level, in: IFIP International Conference on Digital Forensics, Springer. pp. 225–246.

Heeger, J., Yannikos, Y., Steinebach, M., 2021. Exhide: Hiding data within the exfat file system, in: Proceedings of the 16th International Conference on Availability, Reliability and Security, pp. 1–8.

Huebner, E., Bem, D., Wee, C.K., 2006. Data hiding in the ntfs file system. Digital Investigation 3, 211–226. URL: https://www.sciencedirect.com/science/article/pii/S1742287606001265, doi:https://doi.org/10.1016/j.diin.2006.10.005.

Koolhaas, A., van Steenbergen, W., 2020. Apfs slack analysis and detection of hidden data. Security and Network Engineering , 2019–2020.

Larson, S.P., 2009. Concerning file slack, in: Proceedings of the Annual ADFSL Conference on Digital Forensics, Security and Law, Burlington, VT, USA. Peer-reviewed conference paper.

Liu, S.f., Pei, S., Huang, X.y., Tian, L., 2009. File hiding based on fat file system, in: 2009 IEEE International Symposium on IT in Medicine Education, pp. 1198–1201. doi:10.1109/ITIME.2009.5236280.

Mulazzani, M., Neuner, S., Kieseberg, P., Huber, M., Schrittwieser, S., Weippl, E., 2013. Quantifying windows file slack size and stability, in: Peterson, G., Shenoi, S. (Eds.), Advances in Digital Forensics IX. Springer, Berlin, Heidelberg. volume 410 of *IFIP Advances in Information and Communication Technology*, pp. 183–193. doi:10.1007/978-3-642-41148-9_13.

Piper, S., Davis, M., Manes, G., Shenoi, S., 2005. Detecting hidden data in ext2/ext3 file systems, in: Pollitt, M., Shenoi, S. (Eds.), Advances in Digital Forensics, Springer US, Boston, MA. pp. 245–256.

Schwietert, A., Hilgert, J.N., 2025. Data hiding in file systems: Current state, novel methods, and a standardized corpus. Forensic Science International: Digital Investigation 54, 301984.

Srinivasan, A., Rose, C., Wu, J., 2024. slackfs – resilient and persistent information hiding framework. International Journal of Security and Networks 19, 77–91. doi:10.1504/IJSN.2024.140278.

Thampy, R.V., K., P., Mohan, A.K., 2018. Data hiding in slack space revisited. International Journal of Pure and Applied Mathematics 118, 3017–3025.

Toolan, F., Humphries, G., 2025a. Data hiding in symbolic link slack space. Forensic Science International: Digital Investigation 53, 301919. URL: https://www.sciencedirect.com/science/article/pii/S2666281725000587, doi:https://doi.org/10.1016/j.fsidi.2025.301919.

Toolan, F., Humphries, G., 2025b. Data hiding in the xfs file system. Forensic Science International: Digital Investigation 52, 301884. URL: https://www.sciencedirect.com/science/article/pii/S266628172500023X, doi:https://doi.org/10.1016/j.fsidi.2025.301884.

Vangoor, B.K.R., Tarasov, V., Zadok, E., 2017. To {FUSE} or not to {FUSE}: Performance of {User-Space} file systems, in: 15th USENIX Conference on File and Storage Technologies (FAST 17), pp. 59–72.

Appendix A. Supplementary Tables

OS/FS	Cluster Reuse		
	D01A File Deletion	**D01B** Pre-existing Data	**D02** Truncate File
Windows NTFS	✗	✗	✗
macOS APFS	✗	✗	✗
Linux ext4	✗	✗	✗

✗ = no residual data detected (slack cleared/zeroed).

Table A.8: Detection experiment outcomes for standard filesystems.

The following Tables summarize the results of the conducted *Persistence* tests. *Persistence* experiments were only performed for detection scenarios in which Slack data remained recoverable. For Copy-on-Write filesystems, the *-noreflink* option was applied. The rename, filesystem check, and filesystem repair experiments showed no impact on slack data and are therefore omitted. For experiments that trigger *CoW* operations (btrfs and ZFS) the reported results refer to the newly allocated data blocks.

Operation	btrfs	exfat	exfat_fuse	ext2/3/4	fat32	ntfs (fuse)	vfat	vfat_fuse	xfs	zfs
Append 1 Byte	**Clr** d00, d02	**Mix** d00–d02	**Surv** d00–d02	**Clr** d00	**Mix** d00–d02	**Clr** d00, d02	**Mix** d00–d02	**Surv** d00–d02	**Clr** d00	**Surv** d00
Append to Sector	**Clr** d00, d02	**Surv** d00–d02	**Surv** d00–d02	**Clr** d00	**Surv** d00–d02	**Clr** d00, d02	**Surv** d00–d02	**Surv** d00–d02	**Clr** d00	**Clr** d00
Append > Sector	**Clr** d00, d02	**Mix** d00–d02	**Surv** d00–d02	**Clr** d00	**Mix** d00–d02	**Clr** d00, d02	**Mix** d00–d02	**Mix** d00–d02	**Clr** d00	**Clr** d00
Overwrite Start	**Clr** d00, d02	**Mix** d00–d02	**Surv** d00–d02	**Clr** d00	**Mix** d00–d02	**Surv** d00, d02	**Mix** d00–d02	**Surv** d00–d02	**Clr** d00	**Surv** d00
Overwrite End	**Clr** d00, d02	**Mix** d00–d02	**Surv** d00–d02	**Clr** d00	**Mix** d00–d02	**Clr** d00, d02	**Mix** d00–d02	**Surv** d00–d02	**Clr** d00	**Surv** d00
Copy (cp)	**Clr** d00, d02	**Clr** d00–d02	**Clr** d00–d02	**Clr** d00	**Clr** d00–d02	**Clr** d00, d02	**Clr** d00–d02	**Clr** d00–d02	**Clr** d00	**Clr** d00

Legend: Surv = survives, Mix = RAM slack is cleared but drive slack not, Clr = cleared.
Gray IDs indicate the detection scenarios (d00–d02) for which the persistence experiment was executed.

Table A.9: Slack space persistence under Linux across file systems.

Operation	apfs	exfat	fat32	HFS+	vfat
Append 1 Byte	**Clr** d00	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02
Append to Sector	**Clr** d00	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02
Append > Sector	**Clr** d00	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02
Overwrite Start	**Clr** d00	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02
Overwrite End	**Clr** d00	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02
Copy (cp)	**Clr** d00	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00, d02	**Clr** d00– d02

Legend: Surv = survives, Mix = RAM slack is cleared but drive slack not, Clr = cleared.
Gray IDs indicate the detection scenarios (d00–d02) for which the persistence experiment was executed.

Table A.10: Slack space persistence under MacOS across file systems.

Operation	ntfs	exfat	fat32
Append 1 Byte	**Clr** d00	**Mix** d00,d01B,d02	**Mix** d00–d02
Append to Sector	**Clr** d00	**Mix** d00,d01B,d02	**Surv** d00–d02
Append > Sector	**Clr** d00	**Mix** d00,d01B,d02	**Mix** d00–d02
Overwrite Start	**Clr** d00	**Mix** d00,d01B,d02	**Mix** d00–d02
Overwrite End	**Clr** d00	**Mix** d00,d01B,d02	**Mix** d00–d02
Copy (cp)	**Clr** d00	**Mix** d00,d01B,d02	**Clr** d00–d02

Legend: Surv = survives, Mix = RAM slack is cleared but drive slack not, Clr = cleared.
Gray IDs indicate the detection scenarios (d00–d02) for which the persistence experiment was executed.

Table A.11: Slack space persistence under Windows across file systems.