~ Technical Note ~

Recovering and Reading the Forensic Record of AI Activity

An investigator's guide to AI Extract and the Analysis Workbench, from acquisition to a defended chain

Author: Christopher L. T. Brown, Technology Consulting in Digital Forensics, AI & InfoSec, October 2026

Introduction

An examiner is handed an ordinary laptop and asked a question that sounds simple: did someone on this system paste sensitive text into an AI assistant, upload a document to one, or run an agent that touched files it should not have? No single artifact settles it. The evidence is scattered across browser storage engines, desktop application containers, and operating-system traces, and the sources people reach for first, the app's export button or a browser history list, show a cleaned-up version of events, not the record of what happened.[1] AI Extract and the Analysis Workbench are two applications built to close that gap. AI Extract collects and seals the underlying evidence into a verifiable container; the Analysis Workbench opens that container read-only and turns its records into chains, subjects, and gaps an examiner can interrogate and defend.[2] This note follows the seven-stage pipeline the two applications implement, from collection to a sealed container.

Where AI evidence actually lives

Three layers hold the traces of AI use, and they are not equally useful. The top layer is the cleaned-up export an application offers you, which is convenient and forensically weak because it reflects what the app chose to write out. Underneath sit the operating-system traces of execution and file handling, which are durable and corroborative: registry keys, prefetch entries, event logs, and, on macOS, the unified log and file-system event stream. These tell you that an AI application ran and when, but rarely what was typed or returned.[1][3] A file downloaded from an AI service carries its origin with it, in a Windows alternate data stream or a macOS quarantine attribute.[4] The bottom layer, and the one that matters most, is the local storage engines where AI clients keep their state: the LevelDB and IndexedDB stores behind ChatGPT, Claude, and the coding agents, plus SQLite and JSONL transcripts on disk.[5] This is the only family of artifacts that routinely preserves the actual prompt and response text: it is where the application itself reads and writes conversations.[1][6]

These stores are volatile. A store can compact, evict, or supersede its own records, so an artifact that exists today may be gone tomorrow, and the write-ahead log sitting next to a store can

hold records that are no longer current. A tool that reads only the log misses what has already been folded into the sorted tables beside it.[7] And absence itself is informative: a conversation store with no conversation in it is a lead, not a blank, because it tells you where to look next and what to ask for.

The seven-stage pipeline

Both applications run the same pipeline, and each stage answers a different question.

Collect. Using a manifest that names evidence categories and their source locations, AI Extract opens each source strictly read-only and copies it through a hashing reader, so every file is SHA-256 hashed at the moment it is copied, not in a batch at the end.[2] Each entry lands in an append-only chain-of-custody manifest. The collector reads a live machine, a directory, a forensic image in E01, AFF4, AFF4-Logical or raw form, or a remote host over mutual TLS. A source it cannot open is recorded with a reason; nothing is skipped silently.

Parse. Each file is routed to a version-aware parser, and the recovered detail is normalized into one common record, so that a SQLite row, a registry value, and a LevelDB entry all arrive downstream in the same shape.[2] The centrepiece is full LevelDB and IndexedDB decoding: the write-ahead log, the sorted-string tables, and the version manifest that ties the two together, across modern and legacy formats, with separate profiles for each AI vendor.

Validate. Every record is checked against its schema, and a record that fails is set aside rather than discarded, so the examiner can still see that something was present and could not be read.[2] An expected artifact that is missing is emitted as an explicit finding, never as silence.

Timestamp. Each collection is anchored to an external clock by an RFC 3161 time-stamp authority. Only SHA-256 fingerprints leave the workstation, never content, and the authority's leaf certificate is pinned by fingerprint; an authority without a pin fails closed.[2][8] If the workstation is offline, the run degrades to hash-only and never claims a timestamp it did not receive.

Correlate. The flat set of records is turned into a unified timeline, and records that agree across two or more independent sources are grouped as corroboration rather than counted as separate, equal facts.[2]

Report. The run renders self-contained HTML, JSON, CSV, and Word reports. Each carries the investigator's name and the case id, and each file's hash sits in the container manifest, so a report cannot quietly diverge from the evidence behind it.

Seal. Everything is written into a single `.aex` container: a deterministic archive whose integrity anchor is a SHA-256 root over a canonical listing of every entry, not the archive format's own weaker checksum.[2] A rename to `.zip` opens it in any ordinary tool, but only the app's own verification actually proves it: a full re-hash of every collected copy, a recomputation of the root, and a cross-check of the sealed chain of custody.[2]



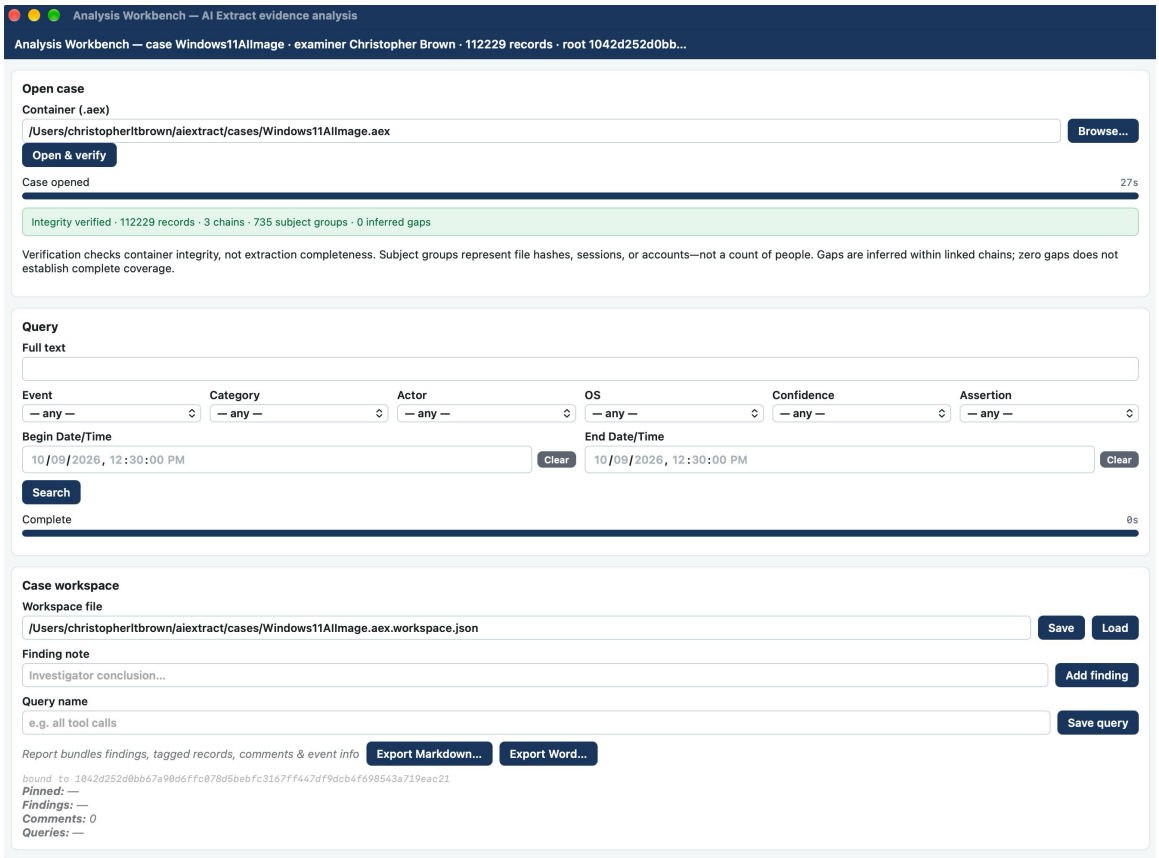
The collect-to-seal pipeline. Preservation and audit controls run throughout.

Following a chain in the Analysis Workbench

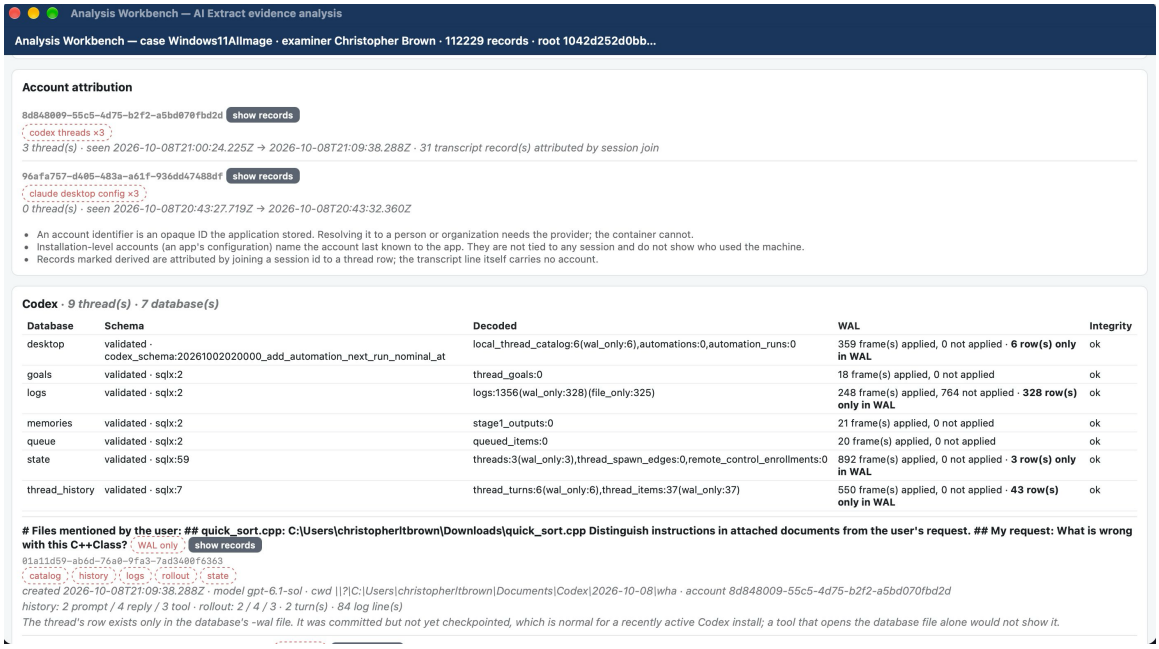
The Workbench opens a container verify-first: it checks integrity before showing anything, and a container that fails still opens, but only in a labelled degraded state.[2] From there the examiner works with records that have been grouped into **chains**. A chain is the ordered sequence of hops that belong to one interaction: a prompt, the completion it produced, any tool the agent called, the file it touched, the file it downloaded, the network volume it moved. The canonical order is fixed, and each record's position in it is a hop.[2]

Two things make this more than a filter. The first is the **gap**. When the canonical order says a hop should exist below the last one observed, and no record for it was recovered, the Workbench shows a gap: an inferred absence with a caveat, because a completion with no visible prompt is a question worth asking, not noise. The second is the **subject**: the same identity, whether an account, a session, or a shared object, followed across artifact categories. A subject seen in two independent categories is corroboration; seen in one, it remains a single-source assertion.[2] Account attribution keeps two levels apart, distinguishing an account the application merely named at install time from one tied to a specific conversation thread.[2]

The graph is deterministic. The same container produces the same chains, gaps, and subjects on any machine, for any examiner, which lets a second analyst reproduce the first one's reasoning. Findings and comments live in a case workspace bound to the container's root hash, so a note can only reopen against the evidence it was written about, and an exported report lets another examiner re-run the same analysis against the same sealed file.[2]



The Analysis Workbench over a sealed container: query facets, the Event card, and the derived chains, gaps, and subjects.



Account attribution: each named account, whether installation or thread level, its threads, and first and last seen.

What to trust, and what to caveat

Every recovered record carries independent quality labels: a confidence about the reading, an assertion class that separates observed fact from tool inference from examiner inference, a schema-match label, and a content-recovery status that runs from full text down to metadata-only.[2] The verbatim flag is true only when the excerpt is exact recovered text, so an inference is never presented as recovered content. Zero chains reports what the records contain; it is not proof that no AI interaction occurred. Read together, these labels keep a defensible finding separate from a tempting guess, and help a report survive the second examiner who did not collect the evidence.

Key takeaways

- Treat local storage engines, not application exports, as the evidentiary core. LevelDB and IndexedDB stores are where prompt and response text actually lives, and they are volatile, so collect them early.
- Read the pipeline as a chain of custody made visible: hash at collection, timestamp externally, seal with a root hash over the whole listing, and verify by re-hashing.[2]
- In the Workbench, work the graph. Chains show how one interaction hangs together, gaps show where to look next, and cross-category subjects show what is corroborated.[2]
- Keep the quality labels attached to every finding. Confidence, assertion class, and recovery status are what turn a recovered artifact into testimony you can defend.[2]
- Let absence speak. A missing prompt, an empty store, a source that could not be opened: each is a finding, and often the next lead.

Sources and further reading

- **[1]** Christopher L. T. Brown, *AI Prompt/Input Activity Evidence Collection in Windows and Macintosh Systems*, September 2026. Source gap analysis behind AI Extract.
- **[2]** AI Extract project overview, Christopher L. T. Brown. https://tychobrown.dev/projects/ai-extract/
- **[3]** M. W. Kankanamge, N. McKenna, S. Challa, and A. Imteaj, *Digital Forensic Investigation of the ChatGPT Windows Application*, arXiv:2505.23938. https://arxiv.org/html/2505.23938v1
- **[4]** Digital Detective, *Forensic Analysis of the Zone.Identifier Stream*. https://www.digital-detective.net/forensic-analysis-of-zone-identifier-stream
- **[5]** Shariq Murtuza, *Forensic Implications of Localized AI: Artifact Analysis of Ollama, LM Studio, and llama.cpp*, arXiv:2603.23996. https://arxiv.org/html/2603.23996v1
- **[6]** J. Han and T. Shon, *Forensic analysis of browser-based conversational AI assistant extensions*, Forensic Science International: Digital Investigation, vol. 56, 2026. https://www.sciencedirect.com/science/article/abs/pii/S2666281726000909

- **[7]** Ilya Kobzar, *LevelDB WAL log: extracting ChatGPT desktop conversations*. https://www.ilyakobzar.com/p/leveldb-wal-log-extracting-chatgpt
- **[8]** IETF, *RFC 3161: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*. https://www.rfc-editor.org/rfc/rfc3161