



BERGSSON FORENSICS

The field handbook.

Read the trace.
Understand the limits.
Explain the finding.



41 artefacts / 10 learning modules / 25 reference digests

FIRST EDITION - OCTOBER 2026

A desk you can take with you.

Scope

This consolidated handbook brings together the current Bergsson artefact catalogue, the ten course lessons and concise maps of the 25 longer field references. Repeated material is organised rather than reproduced word for word. The long references remain available through their linked pages.

Use the evidence, not the label

An artefact is a retained record. It becomes useful when you establish its source, collection context, version and limitations. A tool label, a suspicious name or the absence of a record is not a verdict. Separate presence, capability, execution and actual use.

A practical reading order

- Start with the learning modules on pages 5–14.
- Use pages 15–55 to look up a specific artefact. Each guide includes fields, interpretation limits and a screenshare workflow.
- Use pages 56–80 to choose a longer reference for deeper study. These pages are explicitly digests, not replacements for every original table or code example.
- Return to the website for narration, current sources, the community and quiz badges.

Edition and attribution

Written and compiled from Bergsson Forensics teaching materials. Examples are illustrative and are not allegations about real people. Source links accompany the artefact guides. This edition captures the catalogue on 28 October 2026; Windows behavior, tooling and coverage can differ by version. The subtle watermark identifies the edition without obscuring the text.

[Open the living catalogue](#)



Find your place.

The foundations - pages 5-14

- 05 Start with the evidence
- 06 Reconstruct program execution
- 07 Read the system's recorded activity
- 08 Follow changes through the filesystem
- 09 Investigate behavior beyond a file
- 10 Assess identity, structure, and behavior
- 11 Turn records into a supported finding
- 12 A clear session, from start to finish
- 13 Use the right tool for the question
- 14 Turn several leads into one defensible finding

The artefact desk - pages 15-55

- Programs & execution: 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25
- Files & changes: 26, 27, 28, 29, 30, 40, 41, 54, 55
- User activity: 31, 32, 33, 34, 35, 39, 53
- Commands & scripts: 36, 37, 38
- Startup & persistence: 45, 46, 47
- Devices & software: 42, 43, 44
- Network activity: 48, 49, 50
- Game sessions: 51, 52

The extended desk - pages 56-80

Twenty-five reference digests follow the catalogue order: Prefetch, event logs, NTFS, fileless activity, mod analysis, screenshare workflow, hashes and signatures, Amcache, Shimcache, UserAssist, persistence, Minecraft, FiveM, timelines, reporting, USB, SRUM, processes, memory, PowerShell, ShellBags, shortcuts, MFT, Recycle Bin and browser downloads.

Use the PDF outline to jump directly to any lesson, artefact or reference.

One repeatable method.

01 / Frame the question

Identify the relevant game or application, profile, session and time window. Agree the scope. Record what would support or weaken the hypothesis.

02 / Preserve the starting state

Record time and timezone, Windows build, active processes, launch arguments and source paths. Avoid changing or closing relevant state before documenting it.

03 / Collect deliberately

Keep original paths, names and metadata. Preserve companion databases and logs where required. Record tool names, versions and your collection actions.

04 / Examine and correlate

Use supported parsers on copies. Compare independent sources using object identity and timestamp semantics, not just matching names or nearby times.

05 / State a proportionate finding

Separate observation from inference. Explain alternatives, gaps and confidence. State what remains unestablished and what would resolve it.

06 / Make review possible

Retain hashes and source references. Redact unrelated personal data. Provide reproducible steps and clearly labelled examples.



Start with the evidence

An artifact is a record to interpret. A finding is a conclusion supported by records. Keep those two things separate from the start.

Define the question

Write down what you are trying to establish: whether a program ran, whether a file changed, whether a log was cleared, or whether a mod contains a particular capability. Different artifacts answer different questions. A filename alone cannot establish behavior.

Preserve before you analyze

Record the machine, volume, collection time, timezone, collection method, and source path. Work from collected copies where possible. Calculate a cryptographic hash of collected files so you can check whether the bytes changed. A hash establishes identity and integrity; it does not establish that a file is safe.

On a running system, volatile evidence such as process state and RAM may be lost on shutdown. Collection can itself change the system, so document what you did and follow your acquisition procedure.

Match the artifact to the question

- Prefetch: execution-related records and referenced paths.
- Event logs: recorded system activity, subjects, processes, and events.
- NTFS journals: changes to files and filesystem metadata.
- Memory and persistence artifacts: runtime state and mechanisms that trigger execution.
- Mod contents: claimed identity, code structure, and behavior.

Interpret absence carefully

Missing data can reflect configuration, retention, collection gaps, or deletion. “I did not find a record” is different from “the activity did not happen”. Compare multiple independent sources before making a stronger claim.

Source → observation → interpretation → confidence → limitation. Another analyst should be able to follow the same evidence to your conclusion.

Continue this module online

Reconstruct program execution

Prefetch helps you investigate execution even when the original executable is no longer available.

Know what you are looking at

Windows Prefetch files are commonly collected from C:\Windows\Prefetch. A filename generally combines an executable name and a path-derived hash. That hash is different from a cryptographic hash of the executable's contents. A renamed executable or a different path changes how the artifact needs to be interpreted.

Depending on Windows version and format, a parser can expose execution timestamps, a run count, referenced files, and volume information. Modern formats can retain multiple recent run timestamps, but they are not a complete lifetime history. Format differences and compression make a version-aware parser important.

A practical examination

- Collect the Prefetch files and their filesystem metadata.
- Parse the copies with a suitable tool such as PECmd. Record tool and format information.
- Review the executable name, execution times, run count, paths, and volume references.
- Compare those observations with event logs, files, and other execution artifacts.

Deletion does not remove every trace

A Prefetch file may remain after its corresponding executable is deleted. That gives you an execution-related lead, but it does not recover the deleted executable's bytes or prove that the named program was malicious. Check the path and corroborate the interpretation.

Understand the gaps

Prefetch availability depends on the system's configuration and history. Disabling Prefetch, clearing files, retention limits, and execution techniques can reduce coverage. A run count belongs to the particular surviving artifact; deleting and recreating that artifact can reset the history represented by the new file.

"A matching Prefetch artifact records execution-related data" is supported. "Every action taken by that program is known" is not. Referenced files are context, not a complete audit of all reads and writes.

Continue this module online

Read the system's recorded activity

An event ID is useful only alongside its provider, channel, timestamp, and event fields.

Read the whole record

Collect relevant .evtx logs and note their channels and retention settings. Examine the provider, event ID, timestamp, account or subject, process information, and recorded action. Preserve the original record and use exported views for analysis.

Three useful leads

- Security event 4616: the system time changed. Review the previous and new time, the subject, and the process. Normal clock correction can also produce this event.
- Security event 1102: the Security audit log was cleared. Investigate the subject and surrounding activity; the event records clearing, not its motive.
- System event 7045: a service was installed. Review the service name, image path, and surrounding activity before deciding whether it is suspicious.

Build context around a lead

USB-related events and application crashes can help explain a sequence of activity, but their availability and meaning depend on the provider and environment. Correlate a service installation with its executable, file creation records, and available process telemetry. A crash or a device connection alone is not proof of misuse.

Handle clock changes and missing logs

Normalize timezones and keep the original timestamp representation. If the clock changed, annotate that discontinuity before treating events as a simple chronological list. Missing records may result from retention, configuration, or clearing. Compare other sources rather than assuming that an empty log proves the absence of activity.

When you find a log-clearing event, ask: who performed it, when, what changed nearby, and what independent records survived?

Event definitions: Microsoft: 4616 · Microsoft: 1102.

[Continue this module online](#)

Follow changes through the filesystem

Filesystem records can help connect creation, modification, renaming, and deletion—even when a file is gone.

Two journals, different purposes

\$LogFile is associated with NTFS transaction logging and recovery. Its records can provide context for metadata operations. \$UsnJrnl:\$J is the USN change journal: records describe changes to filesystem objects. Neither should be treated as a complete backup of file contents or as a direct record of program execution.

Read a USN record in context

Review the file reference, parent reference, name, timestamp, and reason flags. File and parent references help you relate records to objects and directories. Reason flags describe classes of changes; they do not identify the full contents of a changed file.

Rename activity can produce old-name and new-name records. Correlate their file references rather than relying only on the visible filename. Use MFT information and other records to help reconstruct the path.

Retention matters

Journals have finite capacity and can wrap. Collection time, volume activity, journal configuration, and tampering affect which records survive. Sequence ordering in a transaction log is not a substitute for a wall-clock timestamp. Do not assume an artifact contains every historical change.

A practical examination

- Collect the relevant NTFS artifacts from the volume using an appropriate acquisition method.
- Parse them with a format-aware tool and retain the source records.
- Group changes by file reference and reconstruct paths where possible.
- Correlate changes with Prefetch, event logs, and the actual files or available copies.

A deletion record supports a filesystem change. To establish that a deleted executable ran, look for separate execution-related evidence.

Further reference: Microsoft: change journal records.

Continue this module online

Investigate behavior beyond a file

When activity depends on memory, scripts, or legitimate tools, a search for an unfamiliar executable is only one part of the investigation.

Start with execution context

Investigate the parent process, command line, user context, network activity, and relevant runtime state. PowerShell and other signed system tools have legitimate uses. Suspicion depends on what they did and how they were invoked, not just their names.

Decode to understand; do not execute to inspect

Encoded or obfuscated commands can hide meaning. Decode and examine them as data in an analysis environment. Encoding alone is not proof of malicious behavior. Preserve the original command and document each transformation so your interpretation is reproducible.

Look for persistence and memory evidence

The source guide covers registry run keys, scheduled tasks, WMI event subscriptions, script-based execution, and process injection. Examine the mechanism, its configured action, its trigger, and the evidence that it executed. Memory can reveal runtime material that does not survive shutdown.

Check which telemetry exists

PowerShell logging, process auditing, and Sysmon provide different views of execution, depending on configuration. For example, Sysmon event 1 records process creation when the relevant telemetry is collected. The absence of a record from a source that was never enabled is not evidence that nothing ran.

A routine administrator script and an attacker's command may both use PowerShell. Compare the command's behavior, parent process, source, destination, persistence changes, and surrounding timeline.

Telemetry reference: Microsoft Sysmon documentation.

Continue this module online



Assess identity, structure, and behavior

A mod's filename is a claim. Its bytes, metadata, code, and behavior are the evidence you use to assess that claim.

Validate identity first

Calculate a cryptographic hash and compare it with a trusted release of the exact same version and build. A mismatch establishes that the bytes differ from that reference. It does not, by itself, establish why they differ or prove cheat functionality.

Inspect what loads and what changes

Review JAR contents, loader metadata, entrypoints, dependencies, and mixin configuration. In Fabric, `fabric.mod.json` describes metadata and can declare entrypoints and mixin configuration files. Trace the referenced classes and methods to understand the implementation rather than stopping at a suggestive name.

Use indicators to guide examination

Strings, package names, embedded resources, and obfuscation are leads. Legitimate mods may contain input, rotation, rendering, or networking code. Evaluate whether the behavior fits the claimed purpose. Static inspection and decompilation can identify functionality; controlled observation can help corroborate it.

Write a defensible finding

Report the source file, hash, relevant class or method, observed behavior, confidence level, and limitations. An unfamiliar or obfuscated mod can remain inconclusive. Do not turn a single keyword match into a confirmed behavioral claim.

Identity: does it match the claimed release? Structure: what code is present and how is it loaded? Behavior: what does that code actually do?

Metadata reference: Fabric: `fabric.mod.json`.

[Continue this module online](#)



Turn records into a supported finding

The strongest account connects multiple kinds of evidence while preserving what each record can—and cannot—establish.

Work a short scenario

You find a deleted executable name in a USN journal, a matching execution-related Prefetch artifact, a nearby clock-change event, and a cleared Security log. Treat these as separate observations first.

- Filesystem: reconstruct the file's changes and identify the object and path.
- Execution: interpret the Prefetch artifact in its version and path context.
- Time: review the old and new clock values, preserve original timestamps, and annotate the discontinuity.
- Logging: investigate the subject of the clearing event and available surviving sources.
- Conclusion: report the supported sequence and identify unresolved gaps.

Keep a timeline with provenance

For each entry, keep the source artifact, record identifier, original timestamp, normalized timestamp, timezone, object or process identity, and interpretation. Match records by more than a filename when stronger identifiers are available.

Separate capability from use

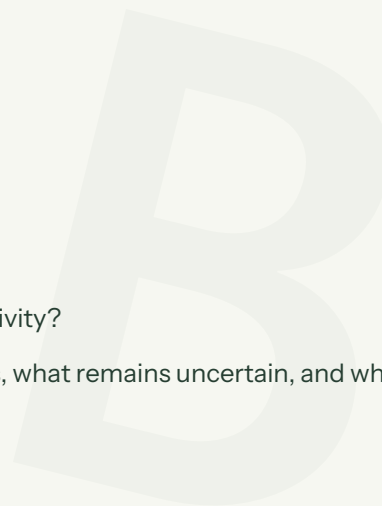
A mod containing a capability does not alone establish when someone used it. A deleted file does not alone establish execution. A cleared log does not alone establish a malicious motive. Explain the distinction and use other sources to address the unanswered question.

Before the final quiz

- Can you select an appropriate artifact for a question?
- Can you distinguish a path hash from a content hash?
- Can you interpret event and journal records in context?
- Can you describe retention and telemetry gaps?
- Can you support a behavioral claim with code or observed activity?

State what was observed, where it came from, what it supports, what remains uncertain, and why your confidence is justified.

Continue this module online



A clear session, from start to finish

A gaming investigation needs a question, a collection plan, and a record another reviewer can follow. This lesson connects those steps into an evidence-first workflow.

Agree the scope before collecting

Identify the game, launcher, profile, mod loader, and relevant session. Explain what you will inspect and where you will keep the results. Keep unrelated personal files, messages, and credentials outside the collection. If the agreed scope is insufficient, pause and clarify it.

Capture the starting state

Record the collection time and timezone, game and Windows versions, active game process, launch arguments, and the relevant installation paths. Multiple launcher profiles can point to different directories. Find the active profile rather than assuming every installation lives in the default folder.

Move through the evidence deliberately

- Document the live state before closing programs or running utilities.
- Collect relevant game logs, configuration, and mod files; retain original names and paths.
- Examine copies. Record hashes, tool versions, filters, and collection gaps.
- Correlate runtime observations with execution and filesystem artifacts from the relevant time window.
- Write the supported finding and unresolved questions. Keep collection actions in the timeline too.

A lead is a reason to investigate

A recently modified mod folder, an unusually sized JAR, or an unfamiliar process can help prioritize examination. None alone establishes cheating. Updates, dependency changes, and launcher activity are alternative explanations. Compare the exact build and inspect behavior before making a stronger claim.

A player has two game profiles and one changed mod folder. Which profile was active? Which file changed? What explains the difference? List the evidence you would need before concluding that an advantage was used.

Continue with the screenshare workflow and live process guide.

Continue this module online

Use the right tool for the question

A tool produces observations. You still need to understand its inputs, coverage, and limitations.

Know the utility you are using

Use the publisher's official project or release page. Record the version, file hash, source URL, and signature information when available. A valid signature identifies a signer and supports an integrity check; it is not a guarantee that every action is safe. A hash identifies particular bytes, not intent.

Choose between live state and history

A process inspection utility can help examine current processes, loaded modules, and open handles. This is a snapshot of what survives at collection time. A process no longer running may require historical execution artifacts. Preserve the live observations with collection time and relevant process identifiers, paths, and context.

Separate configuration from execution

Autoruns reports configured auto-start locations. A listed entry is a configuration lead; separate execution evidence is needed to establish that its action ran at a particular time. Export observations before making changes. Disabling an entry during a check changes the system you are examining.

Keep collection effects visible

Launching a parser, downloading a utility, or opening a file can create new records. Note your actions and timestamps so the next reviewer can distinguish collection activity from earlier activity. Avoid unfamiliar scripts and binaries on the examined machine simply because a handbook names them.

A utility finds a startup entry pointing to a missing executable. Report the configuration, the missing target, and the collection time. Then identify which independent records could answer whether the target ran.

Tool capabilities: Microsoft's Process Explorer, Autoruns, and Sigcheck documentation.

Continue this module online



Turn several leads into one defensible finding

Practice on a fictional case: an unfamiliar JAR, a recent rename, and a game session running at collection time.

Start with three separate observations

The active profile lists a mod. Its JAR differs from the trusted release of the same version. A filesystem record shows a rename. Keep these observations separate: a byte difference, a name change, and an active profile do not yet establish what code ran or what it did.

Test the strongest explanation

Preserve the JAR and reference build, record their hashes, and inspect metadata, entrypoints, mixins, and relevant code paths. Identify the specific behavior implemented and whether it is reachable in this build. Check dependencies and legitimate version differences. Decompiled output can need confirmation against the original bytecode.

Reconstruct the sequence

Correlate the renamed object using file references and paths where available. Compare game logs, launch configuration, surviving execution artifacts, and live observations. Normalize timestamps while retaining original values and annotate any clock changes. State whether the evidence supports presence, capability, loading, or actual use; those are different claims.

Write a reviewable result

Use a compact evidence table: source, collection time, observation, supported inference, and limitation. Include the reference version and hashes, relevant code locations, timeline, alternative explanations, and collection gaps. Redact unrelated personal information before sharing a community finding.

“The collected JAR differs from the reference and contains this reachable behavior” can be supported by the comparison and code. If use during the session remains unestablished, say so and describe the evidence still needed.

Continue with the USN Journal and extended field guides. This scenario is authored teaching material, not an allegation about a real player.

Continue this module online

Prefetch

Windows creates Prefetch records to optimise application startup. A valid record is useful evidence that its associated executable ran; it is not a record of everything the program did.

Location

C:\Windows\Prefetch*.pf

Availability

Coverage depends on Windows edition, configuration and retention. Use a parser supporting the collected format.

Fields to examine

- Executable name and Prefetch identifier; the identifier is not a content hash.
- Internal run timestamps and run count where the format supplies them.
- Referenced files and volume information; referenced files are not automatically executed modules.

What it can establish

- Support execution of an associated executable within the retained history.
- Connect a startup record to a volume and path context.

What it cannot prove

- Identify the human operator or prove cheating.
- Prove that execution never happened because a .pf file is missing.

In a screenshare

- 1. Record the disputed session, timezone and Windows build. Preserve the relevant .pf files before launching investigation tools.
- 2. Parse a copy with PECmd. Inspect internal timestamps instead of treating the .pf file modification time as the complete run history.
- 3. Match path context and time against BAM, a process event, or the actual game logs. Check whether a similarly named legitimate tool explains the record.

Worked example

A launcher-like executable has a Prefetch run timestamp during the session. Report that supported execution observation. Its filename alone does not identify its contents; obtain a matching file hash or another identity-bearing source before attributing it to a particular client.

Cross-check with

BAM, Security event 4688, Amcache

Sources and further reading

PECmd - parser documentation

Guide and full AI narration

Amcache

Amcache is a compatibility and application-inventory registry hive. It can retain file identities and paths after a binary disappears, but inventory is not equivalent to a reliable execution log.

Location

C:\Windows\AppCompat\Programs\Amcache.hve, companion logs and Inventory*.db files where present

Availability

Keys and meanings vary by Windows build. Recent formats may use neighbouring inventory databases; preserve available database journals as well as hive logs.

Fields to examine

- Recorded path, file size, publisher and version when available.
- File identifiers or SHA-1-related fields; their hashing scope can differ from a complete file hash.
- Inventory key times and PE metadata, each with its own meaning.

What it can establish

- Show that Windows inventoried a file or application identity.
- Help connect an old path or filename to a collected program.

What it cannot prove

- Prove execution from an inventory entry alone.
- Use a PE compilation timestamp or registry LastWrite as an exact run time.

In a screenshare

- 1. Acquire the hive with a trusted collection tool; a screenshot of a parser table is not the source hive.
- 2. Parse a copy using AmcacheParser and retain the parser version. Distinguish installed-application entries from individual file entries.
- 3. Compare candidate identities with Prefetch or process events. Explain any partial-hash or build-specific interpretation before comparing hashes.

Worked example

Amcache contains a deleted executable in Downloads. This supports historical inventory of that path. With no independent run artifact, the defensible conclusion is that execution remains unconfirmed, even if the filename resembles a known tool.

Cross-check with

Prefetch, ShimCache / AppCompatCache, Security event 4688

Sources and further reading

AmcacheParser - parser documentation

Guide and full AI narration

ShimCache / AppCompatCache

The application compatibility cache keeps information about executable files Windows has encountered. Interpretation is version-specific, and modern cache presence must not be treated as a universal execution flag.

Location

SYSTEM hive: ControlSet...\Control\Session Manager\AppCompatCache

Availability

The on-disk hive and live cached state can differ. Do not reboot a player machine just to force a cache update.

Fields to examine

- Executable path and parser-supported flags.
- Recorded file modification time, not automatically a run timestamp.
- Control set, Windows build and collection state.

What it can establish

- Support that Windows encountered a particular executable path.
- Provide a path and metadata lead for correlation.

What it cannot prove

- Prove a run time from the cached file modification time.
- Infer execution on every Windows version merely from entry order or presence.

In a screenshare

- 1. Preserve the relevant SYSTEM hive and record whether it was collected live or from an image.
- 2. Parse with AppCompatCacheParser for the source build. Note which control set and entry flags the tool interprets.
- 3. Compare the path with Amcache and Prefetch. Keep the difference between encounter, inventory and execution explicit in the report.

Worked example

The cache lists a utility that the player denies running. It may have been inspected or inventoried. A matching process creation event would materially strengthen the claim; a cache entry by itself does not settle the dispute.

Cross-check with

Amcache, Prefetch, BAM

Sources and further reading

AppCompatCacheParser - parser documentation

Guide and full AI narration



BAM

Background Activity Moderator can record executable activity under a user SID. It is useful for recent execution context, but coverage and cleanup behaviour depend on the Windows version.

Location

SYSTEM hive: Services\bam\State\UserSettings\ ; older layouts differ

Availability

Available on supported Windows 10/11 builds. Not every application or launch mechanism leaves a retained BAM entry.

Fields to examine

- User SID associated with the record.
- Executable path and parsed activity timestamp.
- Hive source, control set and Windows build.

What it can establish

- Support recent executable activity associated with a Windows user SID.
- Correlate a path and time with the disputed session.

What it cannot prove

- Identify the physical person using an account.
- Recover a complete launch history from a record that retains only recent activity.

In a screenshare

- 1. Preserve the SYSTEM hive before deliberately running the candidate executable.
- 2. Resolve the SID using the examined machine's profile information. Parse timestamps with a build-aware registry tool.
- 3. Compare the path and time with Prefetch and process events. State if the record is absent, overwritten or outside the relevant retention window.

Worked example

A BAM value associates a renamed executable with the active profile near the session time. This is useful execution context. Confirm the bytes or program identity separately and avoid presenting the SID as proof of who was physically at the keyboard.

Cross-check with

Prefetch, UserAssist, Security event 4688

Sources and further reading

Velociraptor - Windows execution artifacts

RECcmd - registry parser and batch definitions

Guide and full AI narration

DAM

Desktop Activity Moderator is related to Windows application activity management. Treat its registry records as a distinct, build-dependent source rather than assuming they duplicate BAM on every machine.

Location

SYSTEM hive: Services\dam\...\UserSettings where implemented

Availability

Presence and schema depend on the operating-system build and power-management configuration.

Fields to examine

- User SID and application path where recorded.
- Activity values interpreted by a parser that supports the source build.
- The actual key layout and collection time.

What it can establish

- Provide additional application activity context when supported records exist.

What it cannot prove

- Guarantee the same coverage or timestamp semantics as BAM.
- Treat an absent DAM key as evidence of deleted activity.

In a screenshare

- 1. Record the Windows build and export the relevant hive without enabling services or changing power settings.
- 2. Check the actual DAM key layout and the parser's support. If a value is not documented, preserve it as uninterpreted data.
- 3. Compare with BAM, Prefetch or process events before assigning a run time or user association. Report unsupported interpretations as unresolved.

Worked example

A system has BAM records but no useful DAM entries. The correct result is that DAM supplied no additional evidence for this examination, not that the player removed a history database.

Cross-check with

BAM, Prefetch

Sources and further reading

RECcmd - registry parser and batch definitions

Velociraptor - Windows execution artifacts

Guide and full AI narration

UserAssist

UserAssist records aspects of graphical application use within a Windows user profile. It is particularly helpful for shell-associated launches rather than every possible way to start a program.

Location

NTUSER.DAT: Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist

Availability

Record layouts differ across Windows versions. Encoded names need decoding by a supported parser.

Fields to examine

- Decoded application identifier or path.
- Last-run information and counters available in that format.
- The user profile and registry source containing the entry.

What it can establish

- Support shell-associated application activity in a particular profile.

What it cannot prove

- Cover every service, script, scheduled task or command-line launch.
- Translate focus time or a counter directly into a cheating session duration.

In a screenshare

- 1. Identify the actual profile shown in the screenshare and preserve its NTUSER.DAT hive.
- 2. Parse the UserAssist values rather than treating encoded names as suspicious strings. Retain the timestamp format and tool version.
- 3. Match the decoded path with file identity and corroborate the relevant run time using an independent execution source.

Worked example

A GUI utility appears in UserAssist and Prefetch around the same period. Together they support a shell-associated launch. A missing command-line helper in UserAssist does not contradict a process event showing that helper ran.

Cross-check with

Prefetch, BAM, Security event 4688

Sources and further reading

UserAssist - parser implementation

Guide and full AI narration



Program Compatibility Assistant

Recent Windows builds may retain Program Compatibility Assistant text records of application launches and compatibility handling. These logs can add execution context when the specific file format is understood.

Location

C:\Windows\AppCompat\pca\; for example PcaAppLaunchDic.txt and related files

Availability

Most relevant to supported recent Windows 11 builds. Files and column semantics can change; examine the actual format.

Fields to examine

- Application path and associated text fields.
- Timestamps interpreted with the format's timezone and encoding.
- Which PCA file produced the record.

What it can establish

- Support application-launch or compatibility observations recorded by PCA.

What it cannot prove

- Treat all PCA records as the same event type.
- Assume a missing file means the system was cleaned or never ran a program.

In a screenshare

- 1. Record the Windows build and preserve the original PCA files before sorting or editing them.
- 2. Determine the schema of each file. Use raw lines beside parsed output to validate path and timestamp interpretation.
- 3. Correlate a relevant path with Prefetch or process events and distinguish launches from later compatibility processing.

Worked example

A PCA launch record names a utility at a relevant time, while another PCA entry records compatibility handling. Report the launch and later processing separately; neither record describes what the utility did inside the game.

Cross-check with

Prefetch, Security event 4688

Sources and further reading

Sygnia - original Windows 11 PCA research

Guide and full AI narration



SRUM

System Resource Usage Monitor stores application and resource-usage records. Its intervals can help establish that an application was active and used resources, without providing packet contents or a frame-by-frame session trace.

Location

C:\Windows\System32\sru\SRUDB.dat with applicable ESE companion files

Availability

Requires suitable acquisition of the database and parser support for the source Windows build.

Fields to examine

- Application identities and associated user identifiers.
- Usage intervals and resource counters.
- Network-related totals where present, with their table context.

What it can establish

- Support that an application consumed resources during a recorded interval.
- Add application/user context to a broader timeline.

What it cannot prove

- Prove a specific destination, downloaded file or game action from byte totals.
- Turn an aggregate interval into an exact start and stop time.

In a screenshare

- 1. Preserve the SRUM database with a trusted acquisition method instead of stopping services to unlock it.
- 2. Inspect the relevant table and resolve application/user identifiers using the source data. Record interval granularity.
- 3. Compare usage with process creation, firewall logs or game-session times. Explain coarse intervals and any retention gap.

Worked example

A program used network resources during an interval overlapping a match. That supports resource use, not contact with a particular cheat server. Destination claims require another source that actually records destinations.

Cross-check with

Live network connections, Windows Firewall logs, Security event 4688

Sources and further reading

Velociraptor - examining SRUM

Guide and full AI narration



Windows Error Reporting

Windows Error Reporting stores reports and sometimes dumps for application failures. A crash report can connect an executable, module and failure time, but a crash is not a verdict about malicious behaviour.

Location

ProgramData\Microsoft\Windows\WER\ReportArchive and ReportQueue; Application event log

Availability

Availability depends on reporting policy, cleanup and application behaviour. Dumps can contain sensitive process data.

Fields to examine

- Report.wer application path and report identifiers.
- Failure time and exception or fault-module information where supplied.
- Associated Application error events and dump metadata.

What it can establish

- Support that an application encountered the reported failure.
- Identify a faulting component for further examination.

What it cannot prove

- Infer malicious injection merely because a DLL appears in a crash report.
- Assume a report preserves every module or all earlier executions.

In a screenshare

- 1. Locate a report tied to the relevant executable and preserve the whole report directory.
- 2. Read the application path and failure context alongside Application events; distinguish report creation from the actual failure time.
- 3. Examine a dump only within the agreed scope, then correlate file identity and legitimate overlays or drivers.

Worked example

A game crashed with a capture overlay listed as the faulting module. Record the crash and module identity. A faulting module can be a legitimate installed component and needs examination before any accusation.

Cross-check with

Live processes and loaded modules, Prefetch, Game and launcher logs

Sources and further reading

Microsoft - Windows Error Reporting

Guide and full AI narration



Security event 4688

Security event 4688 records a new process when process-creation auditing is enabled. It can provide a strong connection between an executable, account and creation time within the log's retained coverage.

Location

Windows Security event log; Event ID 4688

Availability

Requires audit configuration and access to the log. Command-line capture is a separate setting and may be absent.

Fields to examine

- New process image and process ID.
- Creator process and subject/account fields according to event version.
- Event timestamp and command line when configured.

What it can establish

- Support process creation recorded by the audited host.
- Correlate parent/child processes and an account context.

What it cannot prove

- Identify executable bytes without a separate identity source.
- Treat no event as proof that nothing ran when auditing or retention was incomplete.

In a screenshare

- 1. Export the relevant Security log without clearing it or enabling a new policy for the historical period.
- 2. Inspect named event fields and version rather than fixed display-column positions. Distinguish hexadecimal and decimal PID representations.
- 3. Correlate PID, image and time together; PIDs are reused. Compare the parent and command line with the actual game launch chain.

Worked example

An event shows a launcher creating a helper process. The creation is supported; the purpose of the helper still requires file identity and context. A missing command line is compatible with disabled command-line auditing.

Cross-check with

Sysmon event logs, Prefetch, Game and launcher logs

Sources and further reading

Microsoft - process creation event 4688

Guide and full AI narration



Sysmon event logs

Sysmon records configured system telemetry such as process creation, network connections and selected file or registry events. What exists depends on both installation history and the active configuration.

Location

Microsoft-Windows-Sysmon/Operational event log

Availability

Sysmon is optional. Filtering can exclude events; installing it now does not recover earlier activity.

Fields to examine

- Event 1 image, command line, hashes and ProcessGuid when recorded.
- Event 3 connection details if network logging is enabled.
- Event-specific fields, host, event time and configuration context.

What it can establish

- Link configured process telemetry and subsequent events through ProcessGuid.
- Support a specific recorded network, file or process event.

What it cannot prove

- Treat all event types as enabled by default.
- Use a signed image or a connection event alone as proof of cheating.

In a screenshare

- 1. Preserve the event log and identify when Sysmon was installed and what configuration covered the session.
- 2. Start from a relevant process event, then join by ProcessGuid where available rather than PID alone.
- 3. Check the exact event semantics before interpreting file-time changes, DLL loads or network activity. Retain benign application explanations.

Worked example

A process creation and network event share a ProcessGuid. This links that process to the recorded connection. It does not reveal encrypted traffic contents or establish that a remote service supplied a cheat.

Cross-check with

Security event 4688, Live network connections, USN Journal / \$UsnJrnl

Sources and further reading

Microsoft – Sysmon events and configuration

Guide and full AI narration



USN Journal / \$UsnJrnl

The NTFS change journal records changes to files and directories on a volume. It is useful for reconstructing create, rename, write and delete activity, including names of files that no longer exist.

Location

NTFS volume metadata: \$Extend\ \$UsnJrnl:\$J

Availability

The journal is bounded and can wrap. Its presence, identifier and retained range are volume-specific.

Fields to examine

- File reference number and parent reference; preserve sequence context.
- USN, record timestamp and reason flags.
- Recorded filename; a full historical path may require reconstruction.

What it can establish

- Support recorded file/directory changes and a sequence within retained journal coverage.
- Connect old/new rename records when file identity and context agree.

What it cannot prove

- Prove file execution, identify the responsible process or recover complete file contents.
- Treat BASIC_INFO_CHANGE as automatic proof of malicious timestamp manipulation.

In a screenshare

- 1. Identify the correct volume. Acquire the journal and useful MFT context before running broad searches that change the live system.
- 2. Parse a copy and inspect reason flags, file references and parent references together. Pair rename events rather than joining on filename alone.
- 3. Build a scoped sequence and compare it with file identity, browser downloads and independent execution records. Document journal wrap or unresolved paths.

Worked example

Records show a file created as update.exe, renamed to helper.exe and later deleted. That supports a file-change sequence. To claim the file ran, add Prefetch or a process event; the journal itself does not supply that evidence.

Cross-check with

\$MFT, NTFS \$LogFile, Prefetch, Browser download records

Sources and further reading

Microsoft - USN change records

Microsoft - change journals

Guide and full AI narration

NTFS \$LogFile

NTFS uses \$LogFile for filesystem transaction recovery. Parsed records can help explain metadata operations and ordering, but this is not a general application, document-reading or execution log.

Location

The \$LogFile metadata file at the root of an NTFS volume

Availability

Requires raw filesystem acquisition and a compatible parser.
The circular log can overwrite earlier transactions quickly.

Fields to examine

- Log sequence numbers and operation/transaction context.
- Referenced file records and parser-supported redo/undo information.
- Metadata changes that can be linked to a specific volume and file identity.

What it can establish

- Support particular filesystem metadata operations in retained transaction records.

What it cannot prove

- Assign a trustworthy wall-clock time to every operation solely from an LSN.
- Prove a program ran or a human deliberately deleted evidence.

In a screenshare

- 1. Preserve \$LogFile together with \$MFT and relevant journal data from the same volume.
- 2. Parse a copy using a tool that supports the format. Keep raw record references beside the interpretation.
- 3. Reconstruct a narrow operation sequence and anchor it to independently timed records. Explain where records are incomplete or only suggestive.

Worked example

A parsed transaction sequence is consistent with a rename seen in USN. It adds filesystem context. Without an independent timestamp anchor, report relative ordering rather than inventing an exact time for every transaction.

Cross-check with

USN Journal / \$UsnJrnl, \$MFT, Directory indexes / \$I30

Sources and further reading

LogFileParser - parser implementation

Microsoft - Master File Table

Guide and full AI narration

\$MFT

The Master File Table describes files and directories on an NTFS volume. It can provide names, attributes, timestamps and record identifiers for current entries and surviving deleted records.

Location

NTFS volume metadata: \$MFT

Availability

Record reuse, filesystem state and acquisition completeness affect recovery. A deleted entry can later belong to a different file.

Fields to examine

- \$STANDARD_INFORMATION and \$FILE_NAME attributes where present.
- Entry number, sequence, parent references and allocation state.
- Size, names and data-stream information supported by the parser.

What it can establish

- Support the presence and recorded metadata of a particular file record.
- Provide identifiers for reconstructing paths and correlating USN events.

What it cannot prove

- Prove execution or identify a person from file presence.
- Treat every timestamp as a user action or recoverable file content.

In a screenshare

- 1. Acquire the MFT from the relevant volume and record the collection method.
- 2. Parse with MFTECmd, retaining entry and sequence identifiers. Compare timestamp sets without assuming every difference is tampering.
- 3. Resolve parent relationships and correlate a suspected rename or deletion with USN, Recycle Bin or other independent sources.

Worked example

A deleted MFT entry retains an executable name. It supports surviving metadata for that record, not necessarily recoverable bytes. Check sequence context before joining it to a recent USN event with the same entry number.

Cross-check with

USN Journal / \$UsnJrnl, Directory indexes / \$I30, Recycle Bin

Sources and further reading

Microsoft - Master File Table

MFTECmd - NTFS parser documentation

Guide and full AI narration

Directory indexes / \$I30

NTFS directory indexes contain entries used to locate children of a directory. Surviving index records or slack can expose historical names and metadata, but stale data needs especially careful attribution.

Location

Directory \$INDEX_ROOT and \$INDEX_ALLOCATION attributes, commonly named \$I30

Availability

Requires appropriate raw acquisition. Index slack and current allocated entries are different evidence classes.

Fields to examine

- Child name and file reference.
- Copied filename metadata within the index entry.
- Directory identity and allocated versus slack context.

What it can establish

- Support a directory/name association represented by a valid entry.
- Provide leads to historical names in surviving slack.

What it cannot prove

- Prove that a slack entry is current or has a complete, trustworthy history.
- Infer execution or content from a directory-name entry.

In a screenshare

- 1. Preserve the relevant filesystem metadata and identify the parent directory.
- 2. Use a parser that distinguishes active entries from recovered slack. Keep source offsets and structural validation.
- 3. Compare file references and names with \$MFT and USN; report stale or unresolved associations separately from current files.

Worked example

A tool-like filename appears only in directory slack. Record it as a historical lead with recovery context. It should not be presented as a currently installed or executed tool without corroboration.

Cross-check with

\$MFT, USN Journal / \$UsnJrnl, NTFS \$LogFile

Sources and further reading

MFTECmd - NTFS parser documentation

Microsoft - Master File Table

Guide and full AI narration

Recycle Bin

Recycle Bin metadata can preserve an original path, file size and deletion context for files moved into the bin. It does not describe every way a file can disappear.

Location

\$Recycle.Bin\ on the relevant volume; paired \$I and \$R items on supported Windows versions

Availability

Permanent deletion, application cleanup and different delete paths may bypass the bin. Retention depends on settings and later use.

Fields to examine

- \$I metadata: original path, original size and recorded deletion time.
- Corresponding \$R data where retained.
- Volume and SID directory containing the pair.

What it can establish

- Support a recorded move-to-Recycle-Bin operation and original path.
- Allow examination of retained file bytes when the data pair survives.

What it cannot prove

- Prove intent to hide evidence.
- Claim that no deletion occurred because the bin is empty.

In a screenshare

- 1. Preserve matching metadata/data pairs and their source volume.
- 2. Parse metadata with RBCmd and resolve the SID context. Confirm the timestamp format and pairing.
- 3. Hash retained data and correlate the file's original path with USN or execution artifacts. Record missing pairs without attempting a live restore.

Worked example

A utility's bin entry predates the disputed match. This supports deletion into the bin at that recorded time. It does not establish that the player ran the utility during the match, nor that all traces were removed.

Cross-check with

USN Journal / \$UsnJrnl, \$MFT, Prefetch

Sources and further reading

RBCmd - Recycle Bin parser

Guide and full AI narration



LNK shortcuts

Windows Shell Link files describe a target and may retain target, volume and path information. The shortcut's own timestamps and copied target metadata describe different things.

Location

User Recent items, Desktop, Start Menu and other locations containing .lnk files

Availability

Shortcuts can be created automatically, copied or made manually. Their meaning depends on origin and context.

Fields to examine

- Target path or shell item identifiers.
- Volume information and working directory where present.
- Shortcut file timestamps versus target timestamps copied into its structure.

What it can establish

- Support a stored shortcut-to-target relationship.
- Add path, volume or recent-item context for a profile.

What it cannot prove

- Prove execution merely because a shortcut exists.
- Treat copied target timestamps as the time the user opened the shortcut.

In a screenshare

- 1. Collect the actual .lnk file and record its location within the profile.
- 2. Parse it with LECmd, separating target fields from the .lnk file's own metadata.
- 3. Correlate relevant Recent items with Jump Lists, UserAssist or program events, and examine whether the link was copied from elsewhere.

Worked example

A Recent shortcut points to a file on a removable volume. It supports a target association within that profile. It does not prove that today's drive letter maps to the same device or that the target was executed.

Cross-check with

Jump Lists, UserAssist, USBSTOR and USB registry records

Sources and further reading

Microsoft - Shell Link format

LECmd - shortcut parser

Guide and full AI narration

Jump Lists

Jump Lists associate applications with recent or frequent destinations and tasks. Automatic and custom destination files have different structures and should not be interpreted as identical histories.

Location

AppData\Roaming\Microsoft\Windows\Recent\AutomaticDestinations and CustomDestinations

Availability

Coverage depends on application integration, Windows settings and retention. An application identifier needs reliable resolution.

Fields to examine

- Application identifier and destination information.
- Embedded LNK target metadata where present.
- DestList fields and ordering only when the format provides them.

What it can establish

- Support an application/destination association in the retained data.

What it cannot prove

- Prove that every listed file was read by a human or executed.
- Treat custom application tasks as a complete chronological file-open log.

In a screenshare

- 1. Identify the user profile and acquire both relevant destination-file types.
- 2. Parse using JLECmd and check whether the row comes from a DestList entry, embedded link or custom task.
- 3. Compare the target identity with LNK files, RecentDocs and application-specific logs. Keep ordering and timestamp semantics separate.

Worked example

An archive utility's Jump List contains a destination matching a downloaded archive. This supports a useful association. Confirm extraction through resulting files or journal changes before saying that the archive's payload was run.

Cross-check with

LNK shortcuts, RecentDocs, Browser download records

Sources and further reading

JLECmd - Jump List parser

Microsoft - Jump Lists

Guide and full AI narration

ShellBags

ShellBags preserve Windows shell view settings and directory-related shell items. They can expose folder or device context after a folder disappears, but do not universally record file reads or exact visits.

Location

User UsrClass.dat and NTUSER.DAT shell BagMRU/Bags structures

Availability

Shell version, profile use and parser support affect what can be reconstructed.

Fields to examine

- Shell item hierarchy and reconstructed directory path.
- Item metadata and key LastWrite with their distinct meanings.
- User hive and source key path.

What it can establish

- Support that a directory was represented in retained shell/profile state.

What it cannot prove

- Prove the user read every file inside a folder.
- Assign an exact first/last visit from every available timestamp.

In a screenshare

- 1. Acquire the relevant user hives and identify the profile being examined.
- 2. Parse with a ShellBags-aware tool, keeping hierarchy and raw key references.
- 3. Correlate a meaningful directory with LNK, Jump Lists or device records. Explain unresolved virtual folders and ambiguous times.

Worked example

A deleted mods folder appears in reconstructed ShellBags. That helps locate historical directory context. It does not identify which mod files were inside at match time; those need independent records or retained contents.

Cross-check with

LNK shortcuts, Jump Lists, \$MFT

Sources and further reading

Velociraptor - ShellBags

Guide and full AI narration



RecentDocs

RecentDocs is a per-user registry source of recently referenced document and shell-item information. It is useful for activity leads but is not an authenticated file-reading audit log.

Location

NTUSER.DAT: Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs

Availability

Collection depends on profile settings, application integration and retention.

Fields to examine

- Recorded document names and shell-item data.
- MRU ordering where available.
- Registry key LastWrite, distinguished from individual item actions.

What it can establish

- Support a retained recent-document association in a profile.

What it cannot prove

- Identify the exact contents of similarly named files.
- Use a parent key LastWrite as the individual open time of every entry.

In a screenshare

- 1. Collect the correct user hive and retain its transaction logs if available.
- 2. Decode the recorded items and distinguish MRU order from timestamps. Preserve the original key path.
- 3. Compare the item with Jump Lists and filesystem identity. Seek a path or file reference rather than joining common filenames alone.

Worked example

RecentDocs names a configuration file. That supports a recent-item lead, not proof that the shown version was edited. Compare the application's own records and the file's change history before attributing modifications.

Cross-check with

Jump Lists, OpenSavePidIMRU / LastVisitedPidIMRU, USN Journal / \$UsnJrnl

Sources and further reading

RECmd - registry parser and batch definitions

Guide and full AI narration



OpenSavePidIMRU / LastVisitedPidIMRU

Common Windows file dialogs retain selected names, shell items or last-used locations in per-user MRU keys. Dialog interaction should be distinguished from a successful file read or save.

Location

NTUSER.DAT: Explorer\ComDlg32\OpenSavePidIMRU and LastVisitedPidIMRU

Availability

Available entries depend on the application, dialog implementation and user settings.

Fields to examine

- Decoded shell item or path and extension grouping.
- Application/location associations where recorded.
- MRU order and key LastWrite context.

What it can establish

- Support that a file-dialog related selection or location was retained.

What it cannot prove

- Guarantee a document was fully opened, saved or executed.
- Assign independent timestamps to entries that do not carry them.

In a screenshare

- 1. Preserve the user hive and parse PIDL structures using a suitable registry parser.
- 2. Record the exact key and distinguish selected item paths from application last-visited folders.
- 3. Correlate with application logs, RecentDocs or created files. Note that cancelled dialogs or different dialog implementations can change interpretation.

Worked example

A file dialog history references an archive on Desktop. It helps focus a search, but successful extraction requires evidence such as output files or a change-journal sequence.

Cross-check with

RecentDocs, Jump Lists, USN Journal / \$UsnJrnl

Sources and further reading

RECmd - registry parser and batch definitions

Guide and full AI narration



RunMRU

RunMRU stores a per-user history associated with the Windows Run dialog. It can identify entered command strings, but it is not a universal command history or a record of successful results.

Location

NTUSER.DAT: Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU

Availability

Depends on use of the Run dialog, policy and retained profile state.

Fields to examine

- Command strings in individual values.
- MRUList ordering.
- Key LastWrite, which is not a timestamp for every stored command.

What it can establish

- Support that the Run dialog retained a particular command string.

What it cannot prove

- Prove that the command succeeded or executed during a particular match.
- Represent PowerShell, terminal and service history as one complete record.

In a screenshare

- 1. Acquire the user hive rather than opening the Run dialog and submitting commands during collection.
- 2. Record strings and MRU order without replacing variables or executing the recorded text.
- 3. Seek a corresponding process event, Prefetch record or resulting file change to test whether a command actually ran.

Worked example

RunMRU contains a command naming a utility. Preserve the string as an input/history observation. A parser must never execute it, and the record alone does not show a successful launch.

Cross-check with

Security event 4688, Prefetch, PSReadLine history

Sources and further reading

RECmd - registry parser and batch definitions

Guide and full AI narration



PSReadLine history

PSReadLine can save commands entered in an interactive PowerShell host. This text history is useful for sequence and intent leads, but normally does not provide reliable per-command timestamps or results.

Location

HistorySavePath from PSReadLine settings; commonly a per-user ConsoleHost_history.txt

Availability

The host, module settings, save style and filtering affect retention. Noninteractive scripts may not appear.

Fields to examine

- Retained command text and ordering.
- The profile, host and configured history path.
- File metadata, distinguished from individual command times.

What it can establish

- Support that particular command text was retained in the examined history file.

What it cannot prove

- Prove command success or exact timing from line order alone.
- Infer no script ran because the history file is empty.

In a screenshare

- 1. Identify the relevant PowerShell host and history path without submitting suspect commands. Preserve a copy.
- 2. Review only the scoped lines; command histories can contain unrelated private information. Never execute history text as part of parsing.
- 3. Correlate selected commands with script-block events, process creation and relevant output files. Keep missing output as an unresolved gap.

Worked example

History contains a download command but no result. A matching browser or filesystem record may confirm a resulting file. Without such evidence, distinguish retained command text from a proven successful download.

Cross-check with

PowerShell script-block logs, Security event 4688, USN Journal / \$UsnJrnl

Sources and further reading

Microsoft - PSReadLine history settings

Guide and full AI narration



PowerShell script-block logs

PowerShell operational events can preserve processed script content when logging was available. Script-block logging and interactive command history are different sources with different coverage.

Location

Microsoft-Windows-PowerShell/Operational or PowerShellCore/Operational; commonly event 4104

Availability

Availability depends on engine version, provider registration, logging configuration and log retention.

Fields to examine

- ScriptBlockId and script text fragments.
- Message numbering for multi-event script blocks.
- Event time, path and process/runspace context where supplied.

What it can establish

- Support that the engine logged the recorded script-block content.

What it cannot prove

- Guarantee every conditional branch ran or the script completed successfully.
- Treat text containing a suspicious function as proof that function produced an effect.

In a screenshare

- 1. Preserve the appropriate engine's log for the existing historical period.
- 2. Reassemble fragments by script-block identity and message numbering. Read the reconstructed text without evaluating it.
- 3. Correlate execution context and any resulting effects with process, file or network sources. Explain disabled logging and retention gaps.

Worked example

A logged script includes a conditional cleanup branch. The logged text supports that the engine handled that block; it does not prove the branch condition was met. Look for independently recorded effects before claiming deletion.

Cross-check with

PSReadLine history, Security event 4688, USN Journal / \$UsnJrnl

Sources and further reading

Microsoft - PowerShell logging

Guide and full AI narration

Browser history

Browser profile databases store visit-related records. They can connect a URL, browser profile and recorded visit context, but automated loads, redirects and synchronisation complicate attribution.

Location

The actual Chrome/Edge profile History database or Firefox places.sqlite, with companion files

Availability

Schemas and timestamp epochs vary by browser and version. Private browsing and retention settings affect coverage.

Fields to examine

- URL, title and visit records supported by the schema.
- Profile identity, visit transitions and source context.
- Timestamp values converted using the correct epoch.

What it can establish

- Support a recorded visit-related entry in the examined browser profile.

What it cannot prove

- Prove the user read a page, downloaded its payload or ran a program.
- Attribute a synchronised or background entry automatically to a local human action.

In a screenshare

- 1. Ask the player to identify the browser and relevant profile before inspecting default folders. Preserve a consistent database copy and available WAL files.
- 2. Use a schema-aware parser or queries against a copy. Validate timestamp conversions against known records.
- 3. Connect a candidate URL to download records and retained files. Limit the examination to relevant hosts and times.

Worked example

A visit record references a tool's release page. That supports recorded browsing context. A successful download and subsequent execution require separate evidence from download state, file identity and a run artifact.

Cross-check with

Browser download records, Zone.Identifier / Mark of the Web, Prefetch

Sources and further reading

Chromium - profile data directories

SQLite - write-ahead logging

Guide and full AI narration

Browser download records

Download tables can retain source URLs, target paths, times and transfer state. These records distinguish a download attempt from an apparently completed transfer; neither automatically means execution.

Location

Browser profile download tables and URL-chain records in the relevant history database

Availability

Fields and status values are browser/version specific.
Preserve the database with its write-ahead-log context.

Fields to examine

- Target or current path and relevant URL chain.
- Start/end fields and completion, cancellation or interruption state.
- Bytes and filename information available in that schema.

What it can establish

- Support a recorded transfer attempt and, when recorded, its completion state.

What it cannot prove

- Prove that downloaded bytes still exist or match a particular release.
- Equate download completion with execution or cheating.

In a screenshare

- 1. Preserve the correct profile's database consistently before making a test download.
- 2. Inspect download state and redirect chains rather than searching filenames alone. Decode times according to that browser.
- 3. Collect and hash the retained file where available, then connect its path to USN, Zone.Identifier or execution evidence.

Worked example

A completed download points to helper.exe, followed by a matching USN rename sequence. This connects transfer and file changes. Add a supported run artifact before concluding the renamed file executed.

Cross-check with

USN Journal / \$UsnJrnl, Zone.Identifier / Mark of the Web, Prefetch

Sources and further reading

Chromium - profile data directories

SQLite - write-ahead logging

Guide and full AI narration

Zone.Identifier / Mark of the Web

Windows may attach origin-zone metadata to a downloaded file in an alternate data stream. A retained stream can add acquisition context but is not a cryptographic identity or a complete download history.

Location

An NTFS file's Zone.Identifier alternate data stream

Availability

Creation and propagation depend on downloader, archive tool, filesystem and Windows version.

Fields to examine

- Zoned and available HostUrl or ReferrerUrl values.
- The exact file carrying the stream.
- Filesystem and acquisition method that preserved alternate streams.

What it can establish

- Support retained zone/origin metadata attached to that file.

What it cannot prove

- Prove execution, harmlessness or authenticity.
- Infer that a file was never downloaded when the stream is missing.

In a screenshot

- 1. Preserve the file using a method that retains alternate data streams.
- 2. Read the stream as text on a copy; do not launch the file or remove its block flag. Record absent fields explicitly.
- 3. Compare origin hints with browser download records and the file hash. Account for archive extraction, copying and non-NTFS destinations.

Worked example

A file has internet-zone metadata and a release URL hint. This is useful acquisition context, but the URL text does not authenticate the bytes or establish who downloaded or ran them.

Cross-check with

Browser download records, \$MFT, Prefetch

Sources and further reading

Microsoft - URL security zones

Guide and full AI narration



USBSTOR and USB registry records

Windows device registry records can identify enumerated USB/storage devices. They help investigate device context, but they do not independently describe every connection or every file transferred.

Location

SYSTEM hive: Enum\USBSTOR and Enum\USB, with relevant device properties

Availability

Device class, Windows version and available properties affect identifiers and timestamps.

Fields to examine

- Device instance, vendor/product and recorded serial information.
- Device properties and their specific timestamp meanings.
- Control set and related storage/volume identity.

What it can establish

- Support that the examined host retained an enumeration record for a device identity.

What it cannot prove

- Prove a particular file was copied or executed from the device.
- Assume a serial string always uniquely identifies one physical device.

In a screenshare

- 1. Acquire the relevant SYSTEM hive and identify device records with a supported parser.
- 2. Compare instance identifiers with SetupAPI records and volume context; avoid using a remembered drive letter as permanent identity.
- 3. Check LNK target volume data or file records before connecting an executable to that device. Record ambiguous identifiers.

Worked example

A USB device record exists, and a shortcut points to a similarly labelled volume. That is a correlation lead. Establish stronger volume identity and actual file activity before claiming a tool was transferred from that device.

Cross-check with

SetupAPI device installation logs, LNK shortcuts, USN Journal / \$UsnJrnl

Sources and further reading

Microsoft - USB storage identifiers

RECcmd - registry parser and batch definitions

Guide and full AI narration

SetupAPI device installation logs

SetupAPI logs record device and driver installation operations. They can help establish when Windows processed a particular device instance, but installation is not the same as every later plug-in event.

Location

C:\Windows\inf\setupapi.dev.log; related setup logs when relevant

Availability

Logs can rotate or retain only part of the period. Interpret time and section status using the actual log format.

Fields to examine

- Device instance identifier and installation section boundaries.
- Operation timestamps and completion/error status.
- Driver package and installation context.

What it can establish

- Support a recorded device/driver installation operation.

What it cannot prove

- List every time a previously installed device was reconnected.
- Prove that the device contained or transferred a specific utility.

In a screenshare

- 1. Preserve the full relevant log, including rotated copies if available.
- 2. Locate the exact device instance and keep the enclosing section, not just a single matching line.
- 3. Compare status and time with USB registry and volume evidence. Distinguish attempted installation from successful completion.

Worked example

SetupAPI records a storage device installation weeks before a match. This supports that installation event. It does not by itself put the device at the match or establish any file transfer then.

Cross-check with

USBSTOR and USB registry records, LNK shortcuts

Sources and further reading

Microsoft – SetupAPI text logs

Guide and full AI narration



Defender detections and history

Microsoft Defender can record detections and remediation actions. These are observations by a security product that require context about the item, action result and detection basis.

Location

Microsoft-Windows-Windows Defender/Operational and applicable Protection History records

Availability

Coverage depends on the active security product, configuration, exclusions and retention. Do not alter protection settings during examination.

Fields to examine

- Threat name and affected resource path where recorded.
- Detection and action events with timestamps.
- Action success/failure and engine/signature context when available.

What it can establish

- Support that Defender reported a detection or attempted a recorded action.

What it cannot prove

- Prove the player executed the file or cheated.
- Treat every detection name as a conclusive independent technical analysis.

In a screenshare

- 1. Export relevant events and preserve the displayed action context. Do not restore quarantined files on the player's live machine.
- 2. Separate detection time from the original file's acquisition or execution time. Check whether remediation succeeded.
- 3. Correlate the affected resource with a retained hash, download record and run artifact. Record the possibility of false positives or unwanted-program classifications.

Worked example

Defender quarantined an archive during download. This supports detection and remediation context; it may specifically mean the payload was prevented from running. Do not convert the detection into a claim of successful execution.

Cross-check with

Browser download records, Prefetch, Security event 4688

Sources and further reading

Microsoft - Defender Antivirus events

Guide and full AI narration



Scheduled Tasks

Task definitions describe configured actions, triggers and security contexts. Task Scheduler events can add creation or run observations when retained; a configured task is not proof that it fired.

Location

C:\Windows\System32\Tasks; TaskCache registry;
TaskScheduler/Operational and relevant Security events

Availability

History availability depends on logging and retention. Task XML, registry state and event history should be examined together.

Fields to examine

- Action executable, arguments, triggers and principal.
- Task path and definition or registration context.
- Run-related events and results where logging captured them.

What it can establish

- Support a task's recorded configuration and specifically logged task actions.

What it cannot prove

- Prove that an enabled trigger fired merely because the definition exists.
- Assume an unusual task name is malicious.

In a screenshare

- 1. Preserve task definitions and relevant history without running the task.
- 2. Resolve the action path, quoting and arguments as data. Identify the account context and trigger conditions.
- 3. Correlate actual run events with process creation and file identity. Consider legitimate updater and maintenance tasks.

Worked example

A task launches a helper at logon. Its definition supports an automatic-start configuration. A run event plus the matching process creation is needed to establish that launch during the examined period.

Cross-check with

Security event 4688, Run and RunOnce keys, Windows services

Sources and further reading

Microsoft – Task Scheduler

Microsoft – Autoruns

Guide and full AI narration

Windows services

Service configuration and service-control events describe installed services and state changes. Installation, starting and successful payload behaviour are different observations.

Location

SYSTEM hive: Services; System service-control events and Security service-installation events where audited

Availability

Event IDs and fields depend on the channel and version. Historical evidence can outlast or differ from current configuration.

Fields to examine

- Service name, binary path, start type and account.
- Recorded installation or state-change event.
- Image identity and relevant event outcome.

What it can establish

- Support an installed service configuration or a specific logged state change.

What it cannot prove

- Prove execution from a registry definition alone.
- Identify a malicious driver solely from an unfamiliar service name.

In a screenshare

- 1. Export relevant configuration and service events before changing service state.
- 2. Inspect the complete binary path and arguments. Resolve environment variables against the examined host rather than the investigator's computer.
- 3. Correlate a service start with process/driver telemetry and exact file identity. Consider legitimate game anti-cheat, hardware and security drivers.

Worked example

A service installation event names a driver near the session. This supports installation of the recorded service. Identify its publisher and bytes and seek load/runtime evidence before alleging activity inside the game.

Cross-check with

Security event 4688, Sysmon event logs, Run and RunOnce keys

Sources and further reading

Microsoft - service installation event 4697

Microsoft - Autoruns

Guide and full AI narration

Run and RunOnce keys

Run and RunOnce registry values describe programs configured for user logon startup. They are configuration evidence; actual execution depends on logon, policy, validity and the specific startup mechanism.

Location

HKCU\HKLM\Software\Microsoft\Windows\CurrentVersion\Run and RunOnce; relevant architecture views

Availability

Different user profiles and registry views can contain different values. RunOnce entries can be transient.

Fields to examine

- Value name and complete command string.
- Hive, user context and registry view.
- Key LastWrite and collected target-file identity.

What it can establish

- Support that a command was configured under a startup mechanism at collection.

What it cannot prove

- Prove the configured target ran at a particular logon.
- Use a key's LastWrite as the individual modification time of every value.

In a screenshare

- 1. Collect relevant user and machine hives without editing startup values.
- 2. Inspect full command strings and targets using Autoruns or a registry parser. Avoid launching suspicious values to test them live.
- 3. Correlate a relevant logon period with process events, Prefetch or task/service evidence, distinguishing configuration from observed execution.

Worked example

A startup value points to a missing executable. This supports a retained startup configuration, which may be stale. The missing target does not establish whether the program ran in the past.

Cross-check with

Scheduled Tasks, Windows services, Prefetch

Sources and further reading

Microsoft – Autoruns

Guide and full AI narration



DNS resolver cache

The local DNS resolver cache can show currently retained name-resolution information. It is volatile system-wide context, not a per-player or per-process browsing history.

Location

Live Windows DNS client cache; inspect with `Get-DnsClientCache`

Availability

Entries expire and can be affected by many applications. Reading the cache is different from collecting historical DNS telemetry.

Fields to examine

- Record name, type and cached answer.
- Remaining TTL and record status where available.
- Collection time and examined host.

What it can establish

- Support that a name-resolution result was in the cache at collection.

What it cannot prove

- Prove a successful connection, download or exact visit time.
- Identify which process or human caused a query from cache content alone.

In a screenshare

- 1. Record the collection time and inspect the existing cache without flushing it or querying suspect domains.
- 2. Preserve the relevant entries and remaining TTL as observed values.
- 3. Correlate with process-scoped DNS telemetry, browser records or network events before attributing the name to an application or session.

Worked example

A release host is cached while a browser is open. It may reflect background activity or an earlier lookup. The cache alone does not show that a tool was downloaded, launched or used in the match.

Cross-check with

Browser history, Sysmon event logs, Live network connections

Sources and further reading

Microsoft - `Get-DnsClientCache`

Guide and full AI narration



Live network connections

A live connection table maps current endpoints and states to owning process identifiers. It provides a snapshot of communication context, not the contents or complete history of a session.

Location

Get-NetTCPConnection, netstat or a trusted live network inspection tool

Availability

Connections can open and close between observations. Permissions and PID reuse affect attribution.

Fields to examine

- Local and remote address/port, state and observation time.
- Owning PID and contemporaneous process identity.
- Creation information only where the source/tool actually supplies it.

What it can establish

- Support an observed endpoint/state association with a process at collection.

What it cannot prove

- Prove what data was transferred or that a shared IP hosts a specific service.
- Treat a later process with the same PID as the earlier owner.

In a screenshare

- 1. Capture the connection table and process identity together before closing applications.
- 2. Resolve the owning image, user, start context and available hash without relying on a friendly process name.
- 3. Compare endpoints with retained firewall/Sysmon events. Explain CDN/shared-hosting ambiguity and encrypted content limitations.

Worked example

A helper process has an established connection to an address used by many services. That supports the observed connection. Identifying a specific remote application or transferred payload requires more than the IP alone.

Cross-check with

Sysmon event logs, Windows Firewall logs, Live processes and loaded modules

Sources and further reading

Microsoft – Get-NetTCPConnection

Guide and full AI narration



Windows Firewall logs

Windows Firewall can log permitted or dropped traffic when the relevant profile and options were configured. These logs provide network event context within that configuration's coverage.

Location

Configured firewall log path; commonly
%SystemRoot%\System32\LogFiles\Firewall\pfirewall.log

Availability

Logging is optional and profile-specific. Rotation, missing fields and action settings limit coverage.

Fields to examine

- Action, protocol and source/destination endpoint fields.
- Recorded time with the log's timezone convention.
- Active profile, logging options and retained period.

What it can establish

- Support a recorded permit/drop event according to the log format.

What it cannot prove

- Read traffic contents or equate a permitted packet with a completed application transaction.
- Prove no traffic occurred when the appropriate logging was disabled.

In a screenshare

- 1. Preserve relevant firewall logs and record their existing configuration without enabling new logging for a past incident.
- 2. Read the header and field order rather than assuming every line has the same schema.
- 3. Correlate matching endpoints and time with live process context or process-scoped telemetry. Keep dropped traffic separate from successfully established connections.

Worked example

A firewall record shows an outbound packet was allowed. It supports that firewall action, not a completed download. Correlate with the application and transfer records before making a payload claim.

Cross-check with

Live network connections, Sysmon event logs, SRUM

Sources and further reading

Microsoft - Windows Firewall logging

Guide and full AI narration



Live processes and loaded modules

Live process and module views describe runtime state visible to the collection tool. They can identify the active game instance and loaded components, subject to permissions and the tool's visibility.

Location

Process Explorer or another trusted process/module inspection view

Availability

The state is volatile. Protected processes or insufficient privileges can limit visibility.

Fields to examine

- PID, image path, command line and process-start context.
- Parent relationships with time and PID-reuse caveats.
- Loaded module paths and exact file identities where available.

What it can establish

- Support that a process or module was visible at the observation time.

What it cannot prove

- Prove malicious injection merely from a DLL's presence.
- Rule out earlier or hidden activity from a clean snapshot.

In a screenshare

- 1. Record the time and capture the current state before restarting the game or stopping processes.
- 2. Identify the actual game path and command line, then inspect relevant modules and identities.
- 3. Check legitimate overlays, recorders, accessibility software and anti-cheat components. Correlate unusual modules with file bytes and other telemetry.

Worked example

An unfamiliar module belongs to a known capture application and its bytes match the installed release. Report that identity and context. A matching name alone would have been insufficient to identify it.

Cross-check with

Sysmon event logs, Windows Error Reporting, Game and launcher logs

Sources and further reading

Microsoft - Process Explorer

Guide and full AI narration



Game and launcher logs

Launcher and game logs help connect the examined files to the actual game instance and session. They can record loaders, mods, versions and errors, but are application-authored and can be incomplete or altered.

Location

The instance path configured by the launcher; for Minecraft, the relevant logs and loader configuration

Availability

Launchers can use multiple profiles or custom directories. Rotated logs and relaunches can replace evidence.

Fields to examine

- Instance path, launcher profile and game/loader versions.
- Recorded session times and reported loaded components.
- Errors, rotation context and links to the examined files.

What it can establish

- Support the application's recorded description of a particular instance/session.

What it cannot prove

- Guarantee the log is complete, authentic or a complete module inventory.
- Treat an empty default mods folder as the state of a different configured instance.

In a screenshare

- 1. Identify the active instance from the launcher and live process context before selecting logs.
- 2. Preserve the relevant log set and configuration before relaunching. Compare reported component versions with actual files.
- 3. Align the session with independent execution or filesystem records, keeping wall-clock and log timezone assumptions explicit.

Worked example

A default mods folder is empty, but the launcher points to another instance whose log names installed mods. Examine that instance. The first folder observation did not describe the game actually used.

Cross-check with

Live processes and loaded modules, USN Journal / \$UsnJrnl, Prefetch

Sources and further reading

Fabric – game log locations

Guide and full AI narration

Thumbnail caches

Windows thumbnail databases can retain rendered previews of items shown by the shell or applications. A cached image can be useful visual context, but mapping it back to a path or a user action requires additional evidence.

Location

User
AppData\Local\Microsoft\Windows\Explorer\thumbcache_*.db

Availability

Cache formats, identifiers and retention vary. Preview generation can be automated.

Fields to examine

- Recovered thumbnail image and cache identifier.
- Database/profile and extraction context.
- A separately established path association, if one can be recovered.

What it can establish

- Support that a preview image was retained in the examined cache.

What it cannot prove

- Prove the user deliberately opened the original file or owned its content.
- Invent an original filename from a thumbnail that has no validated mapping.

In a screenshare

- 1. Preserve cache databases from the relevant profile without browsing unrelated image folders.
- 2. Extract previews with a format-aware tool and retain their cache identifiers.
- 3. Correlate any candidate path with filesystem or shell records; distinguish a visual resemblance from a verified content identity.

Worked example

A cache preview resembles an application screenshot. It is a lead to a rendered image, not proof the pictured application ran on that machine. The image could itself have been downloaded.

Cross-check with

ShellBags, RecentDocs, \$MFT

Sources and further reading

Thumbcache Viewer – project documentation

Guide and full AI narration



Registry transaction logs

Registry transaction logs can help a compatible parser recover a consistent hive state after interrupted writes. They are companions to their source hive, not a complete chronological history of every registry value.

Location

Companion .LOG1/.LOG2 files and other recovery files associated with a registry hive

Availability

Recovery depends on matching hive/log versions and collection consistency. Preserve originals before replay.

Fields to examine

- The source hive and exact matching companion files.
- Sequence/recovery context and parser outcome.
- Recovered values with their provenance.

What it can establish

- Help reconstruct a supported recoverable hive state.

What it cannot prove

- Supply an independent timestamp for every historical registry change.
- Justify combining arbitrary logs with a different hive from another machine or date.

In a screenshare

- 1. Collect the hive and available companion files together and hash the originals.
- 2. Replay only on analysis copies using a compatible parser, recording whether recovery occurred.
- 3. Compare recovered values with unrecovered state and other artifacts. Distinguish transactional recovery from speculative recovery of deleted cells.

Worked example

A parser recovers a recent Amcache value after replaying matching logs. Document that recovery path and source set. The recovered inventory value still does not become proof of execution.

Cross-check with

Amcache, BAM, RecentDocs

Sources and further reading

RECmd – registry parser and batch definitions

Guide and full AI narration



Volume Shadow Copies

Volume Shadow Copies can preserve earlier volume states from which particular files or registry hives may be examined. They offer historical snapshots, not an automatically complete backup or audit trail.

Location

Existing VSS snapshots of the relevant Windows volume

Availability

Snapshots may be absent or outside the needed period.
Application consistency and acquisition method matter.

Fields to examine

- Snapshot identity, source volume and snapshot time.
- The exact path and bytes recovered from that snapshot.
- Comparison with current files or a second snapshot.

What it can establish

- Support the state of an acquired item within the examined snapshot.

What it cannot prove

- Precisely date every change between two snapshots.
- Claim completeness for files or transactions not consistently captured.

In a screenshare

- 1. Inventory existing snapshots without creating or deleting them on the player's machine.
- 2. Acquire only scoped files through a trusted read-only collection method and record snapshot provenance.
- 3. Compare hashes and metadata with current sources.
Describe an interval when a change falls between snapshots, rather than inventing a single action time.

Worked example

An earlier snapshot contains a configuration missing from the current instance. This supports a difference between captured states. Correlate journal and log records to narrow when and why the difference occurred.

Cross-check with

\$MFT, USN Journal / \$UsnJrnl, Registry transaction logs

Sources and further reading

Microsoft - Volume Shadow Copy Service

Guide and full AI narration



Windows Prefetch

When a program ran, where it ran from, and what remains after deletion.

Introduction & Background

This document provides a structured forensic reference for Windows Prefetch files – one of the most valuable and reliable execution artifacts available to investigators on Windows systems.

What Is Prefetch?

Windows Prefetch is a performance feature managed by the SysMain service (formerly SuperFetch in Windows 7 and earlier).

Windows Version Differences

The Prefetch file format has changed significantly across Windows versions. Investigators must identify the correct format version before parsing, as tools calibrated for one version will misparse another.

File Naming Convention

Every Prefetch file follows a precise naming convention that encodes both the executable name and a hash derived from the file's full execution path.

Internal Data Structure

A Prefetch file is a binary structure divided into a header section and a series of data sections. Understanding what each section contains helps investigators know exactly what can be recovered from a .pf file and what questions it can answer.

Timestamps & Run Count

Timestamps are stored in Windows FILETIME format – a 64-bit integer counting 100-nanosecond intervals since January 1, 1601 UTC. They are stored in UTC and must be converted to local time for reporting, keeping the investigator's timezone in mind.

Proving Execution

The most fundamental forensic application of Prefetch is proving that a specific program executed on a specific system. This is particularly valuable when other execution evidence (event logs, AV logs, browser history) has been cleared or was never enabled.

Deleted Program Evidence

One of the most powerful forensic properties of Prefetch is its persistence after the source executable is deleted.

Execution Path Analysis

The execution path embedded in each Prefetch file is one of its most forensically revealing elements. Legitimate system programs consistently run from consistent, known paths.

Accessed Files & Volume Information

Section C of each Prefetch file contains the full paths of every file and directory the executable accessed during the first 10 seconds of its run.

Artifact Locations & Collection

Prefetch artifacts are primarily located in a single directory but must be collected carefully to preserve all associated metadata.

Parsing & Analysis

Prefetch files are binary structures that require dedicated parsing tools. The following workflows cover the most common analysis scenarios from a basic triage through to a full batch investigation with timeline export.

Indicators of Compromise

The following indicators, derived from Prefetch analysis, represent reliable signals of malicious activity. Each can be identified through standard Prefetch parsing without additional tooling.

Anti-Forensics Against Prefetch

Sophisticated attackers are aware of Prefetch and may take steps to defeat it. Understanding these counter-techniques helps investigators recognize when Prefetch has been tampered with and how to compensate using other artifact sources.

Tools Reference

PECmd: CLI Parser: Most comprehensive Prefetch parser; handles all versions including Win10/11 MAM compression; CSV export for timeline analysis: Free – [ericzimmerman.github.io](https://github.com/ericzimmerman)

Conclusion

Windows Prefetch is one of the most reliably available and consistently informative execution artifacts in Windows forensics.

Complete reference and source links

Condensed reference: see the online paper for its complete tables, code and extended examples.

Windows Event Logs

Time changes, cleared logs, driver installations, and the events that connect them.

Introduction to Event Logs in Screenshares

Windows Event Logs are inherently tied to deep system processes, driver creations, memory allocation crashes, and device I/O actions.

Core Event Log Categories

Of the dozens of available logs, four files located in %SystemRoot%\System32\Winevt\Logs\ are used near universally in cheat verifications:

Time Manipulation (4616)

Advanced ghost clients are sometimes hidden among normal files.

Cleared Event Logs (1102 / 104 / 3079)

Users who utilize basic cleaner bat scripts to wipe Prefetch and Temp folders often also run scripts that execute wevtutil.exe cl System , Clear-EventLog , or fsutil usn deletejournal .

Driver and Services (7045)

To bypass anti-cheat protections (such as FiveM's adhesive or server-side Minecraft checks), external clients must map into the game's memory space using a custom Kernel Driver (.sys) or proxy service wrapper.

USB Usage (20001 / 1006)

A classic tactic is for the cheater to place their client on a USB flash drive, load the client on the title screen, inject it, and then rip the USB drive out before entering the SS (Screenshare) voice channel.

Application Crashes (1000 / 1001)

Cheap ghost clients are notoriously unstable. When an injector fails to allocate memory or writes to protected game memory improperly, the game will crash.

Tooling & Triage

During a fast-paced screenshare, manual Event Viewer triage is too slow. Utilizing automated or streamlined search tools is critical for spotting these Event IDs.

Complete reference and source links

Condensed reference: see the online paper for its complete tables, code and extended examples.

\$LogFile & \$UsnJrnl

Reconstruct file activity through transaction logs and the USN change journal.

Introduction

This document provides a comprehensive forensic reference for two of the most powerful – and most overlooked – NTFS artifacts available during a PC check or screenshare: \$LogFile and \$UsnJrnl:\$J.

NTFS Metadata Files - Context

NTFS (New Technology File System) is the filesystem used by all modern Windows installations.

What Is \$LogFile?

\$LogFile is the NTFS transaction log – sometimes called the NTFS redo log. Its primary engineering purpose is to ensure that the filesystem can recover to a consistent state after an unexpected shutdown or crash.

\$LogFile - Structure & Record Types

\$LogFile is organized as a circular buffer divided into 4 KB pages. Each page begins with a standard NTFS update sequence record header (signature RSTR for restart pages, RCRD for log record pages).

\$LogFile - Forensic Value

\$LogFile's primary forensic strength is operating at the transaction level rather than the file level. It records not just that something changed, but the exact before-and-after state of the NTFS metadata.

What Is \$UsnJrnl:\$J?

The Update Sequence Number Journal (\$UsnJrnl) is an NTFS change journal that maintains a high-level, human-readable log of every file and directory operation on the volume.

\$J - USN Record Structure

Each record in the \$J stream is a variable-length structure. Records are always aligned to 8-byte boundaries.

\$J - Reason Flags Reference

The Reason field in each \$J record is a bitmask – multiple flags are often set within a single record, and multiple records may be generated per logical user operation.

\$J - Forensic Value

\$UsnJrnl:\$J is the single most useful file activity artifact available during a screenshare.

Timeline Reconstruction

The most powerful investigative technique combining \$LogFile and \$J is super-timeline construction : merging records from both artifacts (plus MFT timestamps and Prefetch) into a single chronological view of filesystem activity.

Deleted File Evidence

One of the most common defenses during a screenshare is that a suspicious file "was never there" or "must have been from something else." \$J eliminates this defense when the deletion record is still present.

Anti-Forensics & Tampering Detection

A knowledgeable suspect may attempt to interfere with \$LogFile and \$J. Understanding the techniques – and the traces each leaves – is essential.

Gaming & Screenshare Red Flags

In the context of Minecraft and FiveM PC checks, these journals reveal patterns that are specifically meaningful in a gaming anti-cheat context.

Tools Reference

MFTECmd: Eric Zimmermann (EZ Tools): Parses \$MFT, \$LogFile, \$UsnJrnl:\$J, \$Boot, \$SDS. Exports to CSV/JSON.

Conclusion

\$LogFile and \$UsnJrnl:\$J represent two layers of the same truth: a near-complete, tamper-resistant record of everything that happened to the filesystem.

Complete reference and source links

Condensed reference: see the online paper for its complete tables, code and extended examples.

Fileless Malware

Investigate PowerShell, memory, and persistence with little evidence on disk.

Introduction & Background

This document provides a structured forensic and technical reference for fileless malware - one of the most evasive threat categories encountered in modern digital investigations.

What Is Fileless Malware?

Fileless malware is malicious activity that executes entirely within a system's volatile memory (RAM) and does not write a persistent malicious binary to disk.

Attack Technique - PowerShell Abuse

PowerShell is the most frequently weaponized tool in fileless attack chains.

Attack Technique - Registry-Based Persistence

The Windows Registry provides a persistent storage mechanism that attackers exploit to maintain a presence across reboots without placing executable files on disk.

Attack Technique - Scheduled Tasks as Loaders

Windows Task Scheduler allows code execution at login, at defined intervals, on system events, or on network availability.

Attack Technique - Regsvr32 Abuse

regsvr32.exe is a signed Windows binary used to register and unregister COM components.

Attack Technique - WMI Abuse

Windows Management Instrumentation (WMI) is a powerful management framework that attackers exploit for both execution and persistence.

Attack Technique - Process Injection

Process injection is the technique of inserting malicious code into the memory space of a legitimate running process, then executing it from within that process's context.

Encoding & Encryption Techniques

Obfuscation is a foundational component of virtually all fileless malware deployments. Attackers layer multiple encoding and encryption techniques to defeat automated detection, sandbox analysis, and manual investigation.

Forensic Artifacts & Investigation Points

Despite its design intent to leave no trace, fileless malware generates investigable artifacts across multiple data sources. A thorough investigation must systematically examine each of the following.

Memory Analysis

Memory forensics is the cornerstone of fileless malware investigation. Since the payload never reaches disk, a RAM dump captured while the system is still running may be the only source of the actual malicious code.

Indicators of Compromise

The following table catalogues the most reliable IoCs for fileless malware activity. Severity ratings assume no known-good explanation exists for the indicator in the environment being investigated.

Detection Challenges

Understanding why fileless malware evades detection is as important as knowing how to find it. Each challenge below represents a deliberate design decision by attackers - and a gap that defenders must specifically address.

Defensive Strategies & Hardening

Effective defense requires a layered approach. The controls below are listed in approximate order of impact-to-effort ratio.

Tools Reference

Volatility 3: Memory Analysis: RAM dump analysis: processes, injection, networks, strings: Open source - volatilityfoundation.org

Conclusion

Fileless malware represents a sustained and deliberate evolution in attacker tradecraft.

Complete reference and source links

Condensed reference: see the online paper for its complete tables, code and extended examples.

Mod Analysis & Detections

Check authenticity, inspect a mod's structure, and document suspicious behavior.

What Mod Analysis Actually Proves

To analyze unfair mods properly, it helps to think like the player trying to hide one.

Classification of Suspicious Mods

A useful way to classify suspicious jars is to separate visibility from readability. Some clients are obvious and essentially admit what they are.

Trust Validation Before Reverse Engineering

Before pulling classes apart, verify whether the jar is even authentic. A significant amount of time gets wasted reverse engineering files that could have been flagged in the first two minutes by hash comparison alone.

Metadata, Entrypoints & Mixin Review

Loader metadata tells you how a jar introduces itself. Mixin configuration tells you where it plans to interfere.

Static Triage & String Analysis

Static triage is the first real pass through the jar. You are looking for fast indicators that tell you where deeper review should go next.

Detecting Masked or Embedded Clients

Masked clients usually give themselves away through inconsistency. The name says one thing.

Obfuscation & Loader Patterns

Obfuscation changes the reading experience, not the purpose of the code. A cheat still has to read game state, make decisions, alter movement or combat, schedule inputs, render overlays, save config, and activate modules.

Behavioral Analysis in Isolation

Static review gets you far, but some jars only become obvious when loaded in a controlled environment. If you do this step, isolate it.

Reporting Confidence & False Positives

The cleanest reports separate observation from conclusion. Start with what you found, explain why it matters, then state how confident you are.

Useful Workflows & Scripts

The goal of quick scripts is not to replace actual analysis – it is to speed up repetitive work so your attention stays on the jars that deserve it most.

Complete reference and source links

Condensed reference: see the online paper for its complete tables, code and extended examples.



A defensible screenshare workflow

Plan a focused player examination, preserve the evidence, and document the limits of a live view.

Question & scope

A screenshare is a live observation session, not a complete disk acquisition. Start with a concrete question tied to the disputed game session: a particular client, modification, executable, or installation. Agree on the material the player will show and the collection steps before asking for files. Record the game, launcher, operating-system build, account context, and session time. A narrow plan makes the examination easier to reproduce and reduces unrelated browsing.

Screenshots, exported artifacts, and collected files serve different purposes. A screenshot shows what was displayed at that moment; it does not preserve every byte or authenticate the application rendering it. An exported record can be re-examined, but only if its source and extraction method are recorded. Identify which type of evidence supports each observation rather than treating the screenshare itself as proof.

What to collect

Session notes: Start/end time, timezone, stated question, participant consent and scope.

Environment: Game build, launcher profile, instance directory, active user context.

Relevant artifacts: Selected logs, mods, metadata, or execution records associated with the question.

Collection log: Tool names, versions, commands, files copied, and any live changes.

Examination workflow

Ask the player to identify the actual game instance and the disputed session before inspecting a default folder.

Record original paths and available timestamps. Preserve selected files before opening them in tools that may modify data.

Check identity with hashes and exact release versions; then inspect execution or behavior records appropriate to the question.

Connect findings across independently collected sources. Keep uncertain matches separate from confirmed identities.

End with a short observation summary, unresolved gaps, and the evidence needed for further review.

Interpretation & false positives

Launching a parser, opening a folder, restarting a game, or changing a logging option can alter the live system.

Document these actions, because they may create new execution artifacts or rotate logs. A player controlling the displayed view can omit a profile, use a different instance, or show incomplete data without that fact being immediately visible. Record the boundary of what was examined.

An empty folder, a stopped process, or a single clean log is not a clean bill of health for every session. Conversely, an unfamiliar filename is not sufficient grounds for a cheating claim. State the scope of the result: for example, the selected instance contained these files at the time of collection.

Worked scenario

A player shows an empty default mods folder, but the launcher profile points to another instance. Ask for the configured instance path and inspect the corresponding logs and mod files. The first observation described the wrong directory; it did not establish the state of the active game. Preserve the profile information that connects the launcher to the examined path.

Report checklist

The disputed session and actual instance are identified.

Live changes and collection tools are logged.

Each conclusion names its supporting artifact and scope.

Private account details are redacted from shared findings.

Sources & further reading

Velociraptor: file acquisition ↗

Fabric: finding the game logs ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Hashes, signatures & release identity

Compare exact mod and tool releases without confusing a matching hash or signature with safe behavior.

Question & scope

A file's display name is easy to change. Begin a mod, launcher, or utility review by identifying the bytes you actually collected. A cryptographic hash allows you to compare those bytes with an exact reference or a later copy. Record the algorithm, original path, file size, acquisition time, and reference source. For a release comparison, match the version, loader, platform, and build variant—not simply the project name.

Authenticode and content hashing answer different questions. A signature examination reports signing information and validation status for supported files. A content hash is a fingerprint of the collected bytes. Neither tells you everything a program can do or whether its use during a game session was authorized.

What to collect

Collected file: Original path, byte size, acquisition time, and SHA-256.

Reference release: Official release location, version, platform, loader, and expected hash.

Signature result: Status, signer information, timestamp information when available, and tool version.

Comparison notes: Exact files compared and an explanation of any mismatch or unsupported format.

Examination workflow

Collect a copy and calculate its SHA-256 before changing or extracting it.

Retrieve or identify the official reference for the same build through the project's own release channel. Do not substitute a random mirror.

Compare complete hashes, not filename similarity or a short prefix. Retain the reference URL and release description.

For an applicable Windows executable, inspect its Authenticode result separately from its hash comparison.

Investigate mismatches by examining packaging, code, dependencies, and behavior; report what the mismatch establishes before interpreting its cause.

Interpretation & false positives

A mismatch means the compared files differ. It can result from a different loader build, a locally rebuilt project, a changed archive, or malicious modification. Determine which explanation is supported. A matching hash ties the file to the reference bytes; it does not establish that the reference is harmless, that every dependency matches, or that the player used only that file.

A valid signature does not guarantee benign behavior. An unsigned Java archive is not automatically suspicious; Windows Authenticode is not a universal trust mechanism for every file format. Do not treat an unsupported or absent signature as a behavioral finding.

Worked scenario

A Fabric build and a Forge build share a project name but have different hashes. The mismatch is expected until the claimed variant is resolved. After identifying the actual loader, compare the collected file with that exact release. If its bytes still differ, report the difference and locate the changed classes or resources rather than labeling the mod from the hash alone.

Report checklist

The hash algorithm and complete value are recorded.

The reference matches the exact build variant.

Signature and content-identity conclusions are separate.

No capability claim rests on identity checks alone.

Sources & further reading

Microsoft: [Get-FileHash](#) ↗

Microsoft: [Get-AuthenticodeSignature](#) ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Amcache: inventory with context

Use Amcache to identify software and paths while keeping inventory evidence distinct from proof of execution.

Question & scope

Amcache is useful when an examination involves a program that has been renamed, moved, or removed. On relevant Windows systems, the Amcache hive is commonly found under Windows\AppCompat\Programs. Preserve the hive with associated transaction logs when your acquisition method supports them, and record the Windows build. Its structures and field meanings vary across versions, so select a parser that supports the collected format.

Treat the output as a set of recorded application or file observations. Start with identity and path questions: which record refers to the item, what metadata survived, and what other source can establish activity? Avoid translating every inventory entry into a specific launch time.

What to collect

Amcache hive: A collected Amcache.hve and relevant transaction logs.

System context: Windows build, collection method, and acquisition time.

Parser output: Record type, recorded path, timestamps, and available file-identifying metadata.

Related evidence: Matching files, Prefetch, event logs, and filesystem records.

Examination workflow

Acquire the hive and its supporting logs as a coherent collection rather than relying only on a screenshot of registry values.

Parse a copy using a version-aware tool such as AmcacheParser, retaining the original output and tool version.

Search both names and path context. Resolve multiple similarly named records before choosing one.

Compare available identifying metadata with the collected binary or an exact reference, checking how the parser defines each field.

Correlate relevant entries with execution and filesystem artifacts and record which source supports each part of the timeline.

Interpretation & false positives

Inventory can be populated through mechanisms other than the launch you are investigating. The meaning of a timestamp depends on the record type, Windows version, and field definition. A recorded path is valuable context, but the presence of an entry alone does not provide a complete execution history or prove cheating.

If a parser emits a hash-like field, check the implementation and documentation before assuming it covers all bytes or equals a separately calculated cryptographic hash. Preserve the original record identity so another analyst can revisit that interpretation. Missing entries may reflect collection, configuration, version differences, or retention.

Worked scenario

An Amcache entry refers to a suspiciously named utility under a temporary folder. A related Prefetch record and a collected service event are more direct leads for execution context. Report the Amcache path as inventory evidence and discuss the other sources separately. If the file cannot be recovered, do not claim that its name establishes its code or capability.

Report checklist

The Windows build and parser version are recorded.

Inventory, identity, and execution claims are distinguished.

Timestamp meanings are documented rather than assumed.

Matching records have paths and source identifiers.

Sources & further reading

AmcacheParser: author documentation ↗

Eric Zimmerman: tool reference ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

ShimCache without overclaiming

Read AppCompatCache records, interpret their timestamps, and corroborate execution claims.

Question & scope

AppCompatCache, often called ShimCache, is a compatibility-related artifact stored in the SYSTEM registry hive. It can preserve file and path observations relevant to a gaming examination even after an executable is absent from its original folder. The useful question is what the surviving record represents on this Windows build—not whether every listed path must have executed at the displayed time.

Collect the SYSTEM hive with its transaction logs through an appropriate acquisition method. A screenshot of a live cache view loses the surrounding hive context and is harder to reproduce. Multiple control sets may be relevant; preserve which control set the parser used.

What to collect

SYSTEM hive: Collected hive, supporting logs, and control-set information.

Platform: Windows build and acquisition context.

Cache record: Path, record position, available metadata, and parser field definitions.

Corroboration: Prefetch, event logs, collected files, and MFT or journal observations.

Examination workflow

Preserve the hive and logs before interpreting a live tool display.

Use a parser supporting the collected build, such as AppCompatCacheParser, and retain its version and invocation.

Record the control set and full path for every entry used in a finding.

Identify the source and meaning of displayed timestamps; do not relabel a file timestamp as a launch timestamp.

Compare the path and object context with independent execution-related records before proposing a sequence of activity.

Interpretation & false positives

A compatibility record is not a universal execution verdict. Artifact behavior, persistence, and any execution-related indicators depend on the implementation and operating-system version. A recorded file modification time is a property of the observed file, not automatically the moment a user ran it.

Do not assume the order of displayed entries is a reliable chronological launch list. A parser's ordering, control-set selection, and offline-versus-live collection affect interpretation. Absence is equally limited: a missing cache entry does not disprove activity from another well-supported source.

Worked scenario

A cache parser lists a utility with a timestamp before the disputed match. A corresponding Prefetch artifact has a later execution-related timestamp. These values need not conflict: they may describe different events. State the actual meanings, preserve both, and avoid inventing a contradiction by labeling both as launch times.

Report checklist

Control set and operating-system context are available.

The report identifies the actual timestamp field.

Entry order is not presented as a launch history.

Execution assertions use corroborating evidence.

Sources & further reading

AppCompatCacheParser: author documentation ↗

Velociraptor: registry examination ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

UserAssist & user context

Investigate user-specific shell activity without assuming it captures every launch or identifies the person at the keyboard.

Question & scope

UserAssist records are useful for examining user-specific shell activity. The relevant data lives within user registry context, commonly in NTUSER.DAT. Establish which user hive you collected and which profile it belongs to before discussing an application entry. Data formats, counters, and recorded fields vary, so use a parser that explains its interpretation for the collected system.

In a shared gaming computer, the Windows profile, game account, and person operating the machine may be different entities. Keep those identities separate. A user-specific record helps locate context; it does not alone authenticate the human who performed an action.

What to collect

User hive: NTUSER.DAT and relevant transaction logs from the examined profile.

Profile mapping: SID, profile path, and how it relates to the observed session.

Parsed entry: Decoded name, supported timestamps or counters, and raw record location.

Session sources: Game logs, launcher information, and available Windows logon or execution data.

Examination workflow

Identify the actual Windows profile associated with the displayed game instance.

Collect the user hive and logs without modifying registry values to expose a record.

Decode and parse UserAssist using a documented implementation, preserving the raw source and parser output.

Resolve shell identifiers, paths, or aliases with other profile information; do not silently substitute a guessed executable name.

Compare the relevant entry with session time and independent execution records, describing coverage limits in the report.

Interpretation & false positives

UserAssist is not a complete process-start log.

Command-line launches, services, different accounts, and other mechanisms may have different coverage. A parser label such as run count should be interpreted according to the field and version, not assumed to count every lifetime execution.

Decoded names can contain shell-specific identifiers instead of a direct filesystem path. Resolve them cautiously and record the mapping method. An entry in one profile does not establish the state of another profile. Missing data can reflect coverage or collection gaps as well as removal.

Worked scenario

The active game is running from a secondary profile, but a viewer shows UserAssist data for the main account. That output may be genuine and still irrelevant to the disputed session. Collect the correct profile context before comparing entries. If identity remains uncertain, state that limitation rather than assigning activity to the player.

Report checklist

The source hive and profile mapping are recorded.

Decoded names retain their source context.

Counters are not treated as comprehensive lifetime history.

Profile activity and personal attribution are separated.

Sources & further reading

Velociraptor: UserAssist artifact source ↗

RECcmd: registry parser documentation ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Startup, tasks & persistence review

Examine automatic execution mechanisms without treating every startup entry as malicious.

Question & scope

Persistence examination asks what is configured to run again, under which account, and after which trigger. Game launchers, anti-cheat components, update agents, and peripheral utilities often install legitimate automatic execution mechanisms. Inventory an entry's action, path, arguments, account context, trigger, and file identity before drawing a conclusion.

Autoruns can enumerate multiple startup locations. Task Scheduler provides another view of configured tasks. These tools expose configuration; they do not alone establish that an entry executed successfully during the disputed match. Preserve available logs and referenced files to investigate that separate question.

What to collect

Startup entry: Source location, configured action, arguments, and user context.

Task configuration: Task path, triggers, actions, account, and relevant settings.

Referenced file: Content hash, signature information, and exact release context.

Execution sources: Available task, service, process, and game-session records.

Examination workflow

Record the existing configuration before disabling, deleting, or running any entry.

Use a documented viewer and keep complete paths and arguments, including environment-variable references.

Resolve the target file and compare it with the correct vendor or project release.

Separate task registration, configured triggers, and evidence of actual execution.

Correlate relevant activity with session time, collection activity, and the applicable user context.

Interpretation & false positives

A task's name is a label, not an authenticated statement about its purpose. A familiar name can hide a changed target; an unfamiliar name can belong to a legitimate updater. An entry pointing to a missing file may be stale rather than evidence of current capability.

Configured persistence is distinct from use. A task can exist without its trigger firing, and an action can fail. Services and machine-wide tasks may run in a different account context from the player. Avoid attributing every automatic action to an interactive click.

Worked scenario

A task appears to be a launcher updater, but its action points to a user-writable temporary location. Collect the configuration and target bytes before classifying it. Examine file identity, installation context, and available execution records. Report the unexpected target first; explain the behavioral conclusion only if the evidence supports it.

Report checklist

Configuration is preserved before modification.

Names and target identity are assessed separately.

Registration, trigger, and execution are distinct observations.

Account and collection context are included.

Sources & further reading

Microsoft: Autoruns ↗

Microsoft: Task Scheduler ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Minecraft instances, logs & loaded mods

Identify the actual instance and correlate launcher, loader, crash, and mod records for the session under review.

Question & scope

Minecraft players can use multiple launchers, accounts, loaders, and instance directories. Identify the configured game directory and the selected profile before examining a mods folder. An empty default location says little about an instance stored elsewhere. Record the Minecraft, loader, and mod versions relevant to the disputed session.

Fabric documents game logs such as latest.log and archived logs, while crash reports contain crash context rather than the complete normal session log. Preserve the actual files before restarting the game, because later activity can change which session a latest log describes.

What to collect

Launcher profile: Selected instance, game directory, versions, and relevant launch configuration.

Session logs: Current and relevant archived game logs with original paths and times.

Crash context: Associated report, exception information, and session association.

Mod inventory: Collected JARs, configuration, metadata, and exact content hashes.

Examination workflow

Ask the launcher to identify its actual instance folder, and record the selected profile.

Collect the session logs and relevant archives before launching another session.

Read loader and mod-list context with version information; distinguish discovery, loading, and runtime behavior.

Match collected JARs with the instance and exact official release variants, then examine relevant entrypoints and mixins.

Correlate session times with Windows artifacts when the question involves a separate utility or removed file.

Interpretation & false positives

A mod list is not a complete behavioral transcript. A failed startup can discover a mod without reaching a playable session, and configuration can alter its behavior. A log line containing a suggestive class name or mixin target is not sufficient to prove a cheat. Legitimate mods commonly transform game classes.

Logs may contain server addresses, account identifiers, chat, or tokens. Share only the relevant redacted portion publicly while keeping the original evidence appropriately. Local logs can be incomplete or modified, so report their provenance and corroborate important claims.

Worked scenario

A crash report names a combat-related class, but the matching log shows startup failed before the game session began. Examine the JAR to establish capability separately, and avoid claiming that capability was used during the match from the crash alone. If a later successful session is relevant, collect its own records.

Report checklist

The actual instance and loader variant are identified.

Logs are tied to the correct session.

Discovery, loading, capability, and use are separated.

Public excerpts redact unrelated player data.

Sources & further reading

Fabric: locating and interpreting logs ↗

Fabric: mod metadata ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

FiveM client & session triage

Resolve the actual client layout and compare session logs, cache, resources, and Windows artifacts cautiously.

Question & scope

A FiveM examination should start with the actual client installation and session rather than a memorized universal folder layout. Record the client version or update channel, the game installation, relevant client data locations, and the disputed server/session window. Use the active client and its configuration to identify paths, documenting how each location was resolved.

Cfx client documentation and diagnostic reporting guidance are useful references for the current product. Preserve relevant client-generated logs before additional launches or troubleshooting change them. Do not assume that every cache file represents a player-installed modification.

What to collect

Client environment: Installation/data paths, version or channel, and game context.

Session records: Relevant client diagnostic logs with timestamps and original locations.

Resource context: Available resource or cache observations and associated server/session details.

Independent artifacts: Processes, file identity, execution records, and matching filesystem changes.

Examination workflow

Identify the running client's actual installation and relevant data locations rather than scanning one presumed folder.

Collect logs for the disputed session and preserve original path and time information.

Read errors and resource context alongside the client version and session conditions.

Examine unexpected executables or modules through hashes, paths, signatures, and behavior context.

Correlate leads with independent Windows artifacts and describe what cannot be established from client logs alone.

Interpretation & false positives

Servers can deliver resources that appear in client-side context. Cached or resource-related material is not automatically an independently installed cheat. A crash, failed download, or resource error may reflect an ordinary update or server problem. State which observation supports the suspected distinction.

Client layouts and logging change over time. Avoid publishing an unverified list of supposedly forbidden paths as a universal detection method. A local log can be missing or incomplete, and a process name can be changed. Stronger conclusions connect bytes, paths, timing, and observable behavior.

Worked scenario

A cache directory contains unfamiliar script-related names after a server session. Compare the resource context with that session before classifying the files. If a separate utility is suspected, collect its identity and execution-related evidence instead of treating cached server content as the utility. Preserve uncertainty where server context is unavailable.

Report checklist

The actual installation and session are recorded.

Server-delivered resources are considered.

Crash and cache observations are not verdicts.

Windows corroboration is tied to the same time window.

Sources & further reading

Cfx: client manual ↗

Cfx: diagnostic reporting guidance ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Timelines, clock changes & timezones

Build a session timeline that preserves original timestamps and distinguishes ordering from exact time.

Question & scope

A timeline should retain the source and meaning of every timestamp. A file modification value, log timestamp, application-generated time, and parser display can represent different events and use different conventions. Record original values and conversion rules before sorting entries into a single session narrative.

Windows Security event 4616 is a useful source for system-time changes; normal clock correction can also produce it. Examine its old/new time and subject context. A change creates a discontinuity to annotate, not a reason to automatically classify every nearby event as deliberate manipulation.

What to collect

Source timestamp: Original field name, raw value, artifact, and record identifier.

Normalization: Timezone, daylight-saving context, units, and parser conversion method.

Clock events: Relevant time-change records and available old/new values.

Session anchors: Game logs, observed collection times, and independent reference events.

Examination workflow

Define the disputed time window and retain its timezone.

Create separate original and normalized timestamp fields rather than replacing the original values.

Check artifact semantics and resolution; distinguish intervals from point events.

Annotate clock changes and investigate whether they affect the apparent order of nearby records.

Use object and process identities to connect records; do not rely on chronological proximity alone.

Interpretation & false positives

Some sources establish sequence without exact wall-clock time, while others aggregate an interval. Sorting such records by a fabricated precise time can produce a convincing but unsupported story. Prefer a stated interval or sequence constraint when that is all the evidence supports.

A global offset is not automatically valid across a clock discontinuity. Daylight-saving conversion, application-local time, restored files, and archive metadata can all produce differences. Explain unresolved discrepancies instead of silently adjusting timestamps to fit the allegation.

Worked scenario

A download log and a game log appear an hour apart in two viewers. One viewer renders local time and the other UTC. Normalize both using documented rules before describing a contradiction. If a system-time change occurs within the window, preserve separate segments and show which relationships remain uncertain.

Report checklist

Raw and normalized values are both retained.

Timestamp meaning and resolution are recorded.

Clock discontinuities are explicit.

A timeline does not assign unsupported precision or causation.

Sources & further reading

Microsoft: system-time change event 4616 ↗

Microsoft: USN change record context ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Write reproducible findings & tool reviews

Publish useful observations, explain confidence, and submit tools with enough context for an owner review.

Question & scope

A useful community finding lets another investigator reproduce the observation and evaluate the interpretation. Lead with a specific artifact and method rather than a verdict about a person. Record the environment, versions, relevant paths or identifiers, collection method, and the fields that support the claim. Separate the original evidence from any edited screenshot or derived output.

The same standard helps tool review. A project name and download link are not enough to understand what a utility collects, changes, or supports. Submit its official source, intended use, platform, version, limitations, and what you actually tested. A clear submission makes the owner's decision meaningful.

What to collect

Observation record: Artifact, environment, method, relevant fields, and source identifiers.

Reproduction steps: Ordered actions, tool/version, expected result, and observed result.

Interpretation: Supported conclusion, confidence, alternatives, and coverage limits.

Tool submission: Official source URL, platform/version, supported artifacts, license, and collection impact.

Examination workflow

Write the narrow question and describe the evidence before stating the interpretation.

Preserve original files or records privately and make redacted excerpts for public discussion.

Include enough steps and versions to reproduce the method, plus differences from the reference environment.

State alternative explanations and identify the extra evidence needed to distinguish them.

For a tool, verify the project's source location and document your test conditions without implying owner approval.

Interpretation & false positives

A public display name does not verify authorship, and a community post is not automatically reviewed research. Evaluate the underlying method and evidence. A screenshot can be useful illustration, but it cannot replace the original record for byte-level identity or deeper parsing.

Approval in a tool directory means the owner reviewed that submission; it is not a perpetual guarantee for future releases or every download mirror. Record the reviewed version where practical and use the official project source. If a claim cannot be reproduced, report that limitation rather than quietly upgrading its confidence.

Worked scenario

A post claims one filename always proves cheating. Reproduce the query in a known legitimate instance and compare the artifact's actual meaning. Share the counterexample and explain why filename matching overstates the finding. A constructive correction is more useful than replacing one unsupported verdict with another.

Report checklist

Observation and interpretation are distinct.

Versions and reproduction steps are included.

Player identifiers and secrets are removed from public excerpts.

A suggested tool is not described as approved until the owner decides.

Sources & further reading

NIST SP 800-86: forensic methodology ↗

Microsoft: content integrity hashing ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

USB devices & removable paths

Relate installation, connection, and file-path observations without turning a device record into a transfer log.

Question & scope

Removable storage can explain a historical drive letter, a shortcut target, or a utility that is no longer present on an internal volume. Ask which device and which connection window matter to the disputed session. Windows installation logs and registry or event context can provide identifiers, but installation, connection, mounting, and file transfer are different events.

SetupAPI.dev.log records device installation information and commonly resides in the Windows INF directory. Preserve a collected copy before examining it. Installation records are useful context, but they do not enumerate every subsequent insertion or every file copied from the device.

What to collect

Installation log: A collected SetupAPI.dev.log with original metadata.

Device identity: Available instance identifiers, serial information, and source fields.

Volume context: Historical volume information, mount context, and relevant drive letters.

File evidence: Shortcuts, Prefetch volume references, and filesystem or game records.

Examination workflow

Record the Windows build and the device-related question before choosing sources.

Collect installation, registry, and relevant event context; document which sources were available.

Match complete identifiers cautiously. A generic device class or display name is not a unique identity.

Resolve historical drive letters using volume context before assigning a path to a current device.

Correlate device context with specific file and execution records, keeping unsupported transfer claims separate.

Interpretation & false positives

A device installed once may produce no new installation record during a later insertion. Event availability varies across builds, providers, and logging configuration. Do not assign one universal event ID or a fixed retention period to every system.

Serial and instance fields may represent different layers of identity; some devices provide weak or non-unique identifiers. A connected drive is not proof that a particular file was copied or run. An absent drive at collection time does not eliminate surviving historical paths.

Worked scenario

A shortcut and a Prefetch volume reference point to a removable volume, while installation data identifies a compatible USB device. This is a correlation lead, not a complete transfer history. Record how the volume and device were mapped, identify uncertainty, and search for the specific file's independent records.

Report checklist

Installation and later connection are distinguished.

Device and volume identifiers retain their source fields.

Drive letters are interpreted historically.

Transfers or execution are supported by file-specific evidence.

Sources & further reading

Microsoft: SetupAPI device installation logs ↗

LECmd: link and volume context ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

SRUM: resource use in context

Use resource and network-usage records as timeline context without treating them as packet captures or exact launch logs.

Question & scope

System Resource Usage Monitor data can help investigate application activity and resource usage. Its Windows ESE database is commonly located at `Windows\System32\sru\SRUDB.dat`. Depending on collected tables and system version, records can provide application, user, resource, or network-usage context. Establish the available schema and parser interpretation before relying on a field.

Normal launchers, voice applications, overlays, and updates can produce substantial activity. Compare the recorded identity and interval with the relevant session instead of equating high usage with misconduct.

What to collect

SRUM database: Collected `SRUDB.dat` and supporting ESE files required by the acquisition method.

Schema context: Windows build, available tables, and parser version.

Identity context: Application identifiers and associated user or mapping information.

Time context: Original timestamps, aggregation intervals, and matching session records.

Examination workflow

Use an acquisition method that produces a coherent database copy rather than stopping services merely to unlock the file.

Select a parser supporting the source schema and preserve its table-level output.

Resolve application and user identifiers using relevant mappings; document unresolved identifiers.

Review the interval represented by each resource or network record before comparing it with a match.

Correlate leads with process, event, file, and game records rather than treating the output as a verdict.

Interpretation & false positives

Usage accounting is not packet content. A network-usage figure does not identify every destination or disclose what bytes were exchanged. Aggregated records should not be relabeled as precise program-start or transfer timestamps.

Collection state, parser support, configuration, and retention limit coverage. A missing application record is not a universal disproof of execution. Shared processes, services, launchers, and legitimate downloads can make a suspicious-looking pattern ordinary once identity and interval are resolved.

Worked scenario

A launcher records substantial network use near a disputed session. Its update log and filesystem records show a game update. Evaluate that explanation before describing the usage as a tool download. If a process cannot be identified confidently, record the unresolved mapping and avoid assigning its bytes to a player action.

Report checklist

Database collection and schema support are documented.

Mappings are resolved or explicitly uncertain.

Accounting intervals are not presented as exact event times.

Network usage is not described as packet content.

Sources & further reading

Microsoft Security: forensic SRUM guide ↗

Velociraptor: SRUM examination ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Live process & module inspection

Inspect process identity, parents, modules, and open resources without equating overlays or unusual names with cheats.

Question & scope

A live process view can establish what is running during an observed session. Record collection time and tool version before discussing a PID, path, command line, parent, user, module, or handle. Process IDs can be reused after termination; a PID without time and identity context is not permanent.

Process Explorer can expose relationships, loaded DLLs, and handles depending on access. Use those views to form a specific question. An overlay library, recorder, accessibility application, or security component may legitimately interact with a game process.

What to collect

Process identity: PID, collection time, image path, user, and available start information.

Invocation context: Command line, parent identity, and relevant environment.

Modules and handles: Available module paths and resource references with access limits.

File identity: Collected hashes and signature information for relevant binaries.

Examination workflow

Capture observed state before stopping processes or relaunching the game.

Resolve the executable path and parent context rather than trusting a display name.

Inspect relevant modules and resources, retaining path and identity information.

Compare suspicious modules with exact releases and legitimate software in the environment.

Correlate runtime observations with logs and files, documenting views denied by access restrictions.

Interpretation & false positives

A live tool displays a moment in time. A module can load or unload between observations; a terminated process may be absent. A failure to inspect a process can reflect privilege or protection rather than concealment. Do not turn a blank panel into proof that no relevant state existed.

A DLL inside a game process is not automatically maliciously injected code. Names, signatures, and paths are identity signals, but capability and use require further examination. Parents may have exited, and current PID mappings can differ from earlier records.

Worked scenario

An unfamiliar DLL appears in the game process. Its hash and publisher information match an installed capture product, and notes confirm recording was active. Record the supported explanation. If the bytes differ from the expected release, investigate that discrepancy instead of treating every similarly named module alike.

Report checklist

Each PID is paired with time and identity.

Access restrictions are documented.

Module presence is separate from a behavior claim.

Legitimate overlays and capture products are considered.

Sources & further reading

Microsoft: Process Explorer ↗

Microsoft: Sysmon telemetry ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Memory triage for gaming processes

Plan memory collection and interpret process-scanning leads with reproducible acquisition context.

Question & scope

Memory may preserve process state, loaded material, strings, and runtime context absent from disk. Decide whether collection is appropriate before asking a player to run an acquisition tool. Record its provenance, version, required access, output location, and impact. A live acquisition is a changing-system operation, not an invisible snapshot with no effect.

Separate acquisition from analysis. Preserve the image and calculate a content hash. Record the machine, Windows build, collection start and end times, and errors. Analyze a copy in a controlled environment; do not execute material found within it.

What to collect

Memory source: Image, content hash, method, and collection times.

System context: Windows build, machine identity, available symbols, and acquisition limits.

Process observations: Parser-supported identity, tree, command-line, or scanning results.

Corroboration: Relevant files, logs, modules, and session context.

Examination workflow

Define the process or capability question and select a compatible acquisition method.

Preserve output and errors; do not substitute a process dump for complete RAM without stating the difference.

Use a documented framework such as Volatility 3 with suitable symbols and record commands and version.

Compare process enumeration with scanning results where appropriate, investigating differences rather than assuming every object is active.

Treat suspicious regions or strings as leads requiring process, mapping, file, or behavioral corroboration.

Interpretation & false positives

Enumeration and pool scanning can answer different questions. Scans can encounter residual objects; a difference from a live list does not itself prove a hidden active process. Symbol mismatches, acquisition gaps, and operating-system support affect results.

Unusual or executable regions are not automatically cheat evidence. JIT runtimes, security products, graphics components, and legitimate hooks can create complex layouts. A string may be cached text rather than an executed capability. Retain its offset and process context, and describe interpretation limits.

Worked scenario

A scan contains a process name absent from the live list. Check whether its lifetime overlaps the session and whether the object is residual. Compare creation or exit context and other artifacts. Report a historical or unresolved object unless evidence supports current activity.

Report checklist

Acquisition type, hash, and timing are recorded.

Framework and symbol support are documented.

Residual objects are distinguished from active processes.

Strings or region flags are not complete capability proof.

Sources & further reading

Volatility 3: process-list documentation ↗

Volatility 3: framework source ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

PowerShell logs & command context

Read script telemetry and interpret encoded commands without executing them.

Question & scope

PowerShell can support administration, launcher maintenance, collection, or malicious activity. Establish what telemetry was enabled and which engine produced it. Windows PowerShell and newer installations can use different provider or channel contexts. Collect the actual logs instead of assuming one default location covers every invocation.

Preserve original text and record identifiers before decoding or reformatting a command. Document every transformation. A script record is useful because it preserves observed content or context, not because a long or encoded string is automatically suspicious.

What to collect

Relevant EVTX: Provider, channel, retention, and collection time.

Script records: Text, available identifiers, fragments, and timestamp fields.

Process context: Parent, command line, user, engine version, and telemetry.

Related state: Referenced files, tasks, registry changes, and network observations.

Examination workflow

Record the engine and existing configuration; new logging cannot create historical evidence.

Collect original logs and inspect record fields rather than searching screenshots for a keyword.

Group fragmented content using available identifiers and note missing portions.

Decode data offline without invoking the recovered command; retain original and derived text.

Interpret behavior using process and user context, then seek corroborating file, persistence, or network records.

Interpretation & false positives

Coverage depends on configuration and engine behavior. Missing script text from an unconfigured source does not prove no script ran. A command may contain unavailable variables or incomplete content; its outcome needs support beyond the recorded text.

Encoding and downloaded text can have legitimate purposes. A collector can generate PowerShell artifacts during a screenshare. Compare timestamps with the collection log so that your own tooling is not attributed to earlier activity.

Worked scenario

An encoded command appears after a collector was launched. Notes identify it as an approved acquisition step. Preserve that attribution and examine earlier relevant records separately. A different earlier command deserves its own interpretation rather than being grouped with every encoded invocation.

Report checklist

Engine, provider, channel, and coverage are recorded.

Original and decoded text are retained.

Fragments and unresolved variables are identified.

Collector activity is separate from earlier activity.

Sources & further reading

Microsoft: PowerShell logging ↗

Microsoft: Sysmon context ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

ShellBags & directory context

Reconstruct user-specific directory context while distinguishing folder navigation from file execution.

Question & scope

ShellBags can preserve information about directories encountered through the Windows shell. During a game examination, they may provide context for a removed client folder, an external drive, or a different instance location. These records relate to shell presentation and directory context; they are not a complete record of opening every file inside a folder.

Collect the relevant user registry hives, including `UsrClass.dat` where applicable, with supporting transaction logs. Preserve the Windows profile association. A parser may reconstruct a directory hierarchy from multiple records, so retain the records and reconstruction context instead of relying solely on the final displayed path.

What to collect

User hives: Relevant `NTUSER.DAT` and `UsrClass.dat` copies with supporting logs.

Profile identity: SID, profile path, Windows build, and acquisition time.

Directory records: Parsed hierarchy, shell-item information, and source key paths.

Supporting paths: LNK targets, Jump Lists, filesystem records, and game instance configuration.

Examination workflow

Locate the hives for the profile actually examined rather than assuming the first profile is the active player context.

Parse collected copies using a documented ShellBags implementation and record its version.

Follow the reconstructed directory hierarchy and retain any unresolved or incomplete segments.

Compare external-volume or removed-folder references with device, shortcut, and filesystem evidence.

Report which directory context survived and which further claim would require separate evidence.

Interpretation & false positives

Directory context does not establish that a particular executable ran. It may help explain where to look, but the contents of a directory can change after the record is created. Different timestamp fields may originate from shell items or registry keys and describe different things. Do not collapse them into a single last-opened time without verifying the field semantics.

A reference to removable storage is not a complete history of transfers. An absent ShellBag is not proof that a folder was never visited. Collection coverage, application behavior, profile selection, and artifact retention all limit the result.

Worked scenario

A removed folder name appears in reconstructed shell context and a shortcut points to a similarly named executable. This supports a path lead. Compare volume and object information before treating the names as the same location, then look for independent execution-related records. The directory record itself should remain a directory observation in the report.

Report checklist

The profile and both relevant hive sources are considered.

Reconstructed paths identify unresolved segments.

Each timestamp is reported with its originating field.

Directory context is not presented as executable use.

Sources & further reading

Velociraptor: ShellBags parser ↗

Eric Zimmerman: ShellBags tools ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Shortcuts & Jump Lists

Connect recent items and target paths to the relevant application, profile, and volume.

Question & scope

Windows shortcuts and application Jump Lists can provide target paths and user-specific context for recent items. They can be useful when a file has moved or disappeared, or when a game instance was launched from an unexpected directory. A shortcut describes a target and may retain metadata from the time it was created or updated. It is not a complete record of the target's subsequent history.

Jump Lists are attached to applications and can include recent, frequent, pinned, or application-defined destinations and tasks. Identify the application association and profile before drawing conclusions from a parsed entry. Pinned or static tasks can persist without representing recent use of the target.

What to collect

Shortcut files: Selected .lnk files, source locations, and filesystem metadata.

Jump List files: Relevant user-profile destination lists and their metadata.

Target information: Path, working directory, arguments, volume context, and parser-supported fields.

Application context: The associated application or AppID and how that association was determined.

Examination workflow

Collect files from the profile and instance context relevant to the session.

Parse shortcuts and destination lists using format-aware tools such as LECmd and JLECmd.

Keep the shortcut file's own timestamps separate from embedded target metadata.

Resolve drive letters using available volume information; a historical drive letter may not name the same current device.

Compare targets and application context with game logs, filesystem changes, and independent execution artifacts.

Interpretation & false positives

A target name appearing in a link does not prove its current presence, contents, or execution. A manually created shortcut can refer to a file that was never launched. A recent destination may describe a document opened by an application, not a separate process execution.

Historical paths can contain usernames and private directory names; redact those from a public finding while retaining an unredacted evidence copy where appropriate. Exact interpretation requires distinguishing the source file, the application's destination entry, and the target object.

Worked scenario

A shortcut names a client on drive E:, but the current E: device is a different USB stick. Compare volume identifiers and accompanying device records before attributing the target to the connected stick. Report the historical target context as recorded and leave uncertain mappings unresolved.

Report checklist

Source and embedded target timestamps are separated.

Application and profile associations are stated.

Historical drive letters are not treated as permanent identities.

Execution claims rely on more than a shortcut.

Sources & further reading

Microsoft: Jump List behavior ↗

LECmd: shortcut parser ↗

JLECmd: Jump List parser ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

The MFT: file identity & metadata

Trace file records, names, and timestamp families without assuming metadata reveals behavior.

Question & scope

The NTFS Master File Table describes files and directories on a volume. In a gaming investigation, it can help establish that an object existed at a recorded path, identify related metadata, and connect a removed client to surviving filesystem records. An MFT record is filesystem evidence, not a direct observation of what a program did.

Collect the relevant volume context and use a parser appropriate to the acquisition. Keep the original record identifiers and sequence information where available. After deletion, a record can become available for reuse; a matching record number alone may therefore be insufficient to identify the same historical object.

What to collect

Filesystem source: Collected MFT or disk image with volume identity and acquisition notes.

Record identity: Entry number, sequence information, parent reference, and allocation state.

Names and attributes: Parsed filename attributes, data descriptions, and relevant timestamp families.

Related sources: USN journal, transaction log, actual file copies, and execution-related evidence.

Examination workflow

Identify the volume and acquire its metadata through a method suited to the task.

Parse a copy with a documented tool such as MFTECmd, recording the version and input format.

Search full path context and follow parent references instead of assigning identity from a filename alone.

Keep each timestamp family and source field separate. Describe any discrepancy before interpreting it.

Correlate file creation, rename, or deletion leads with journal and execution records; retain raw identifiers in your working notes.

Interpretation & false positives

File metadata can change for legitimate reasons such as copying, extracting archives, updates, and normal application activity. Timestamp differences do not automatically establish deliberate tampering. A date printed by a parser is meaningful only with its attribute, field, and conversion rules.

Microsoft describes deleted records as reusable. Old metadata may survive temporarily, but it is not a guaranteed history or a guarantee of recoverable content. Do not assume a freed record still describes the file currently carrying a similar name. Execution and user attribution require additional sources.

Worked scenario

An archive was extracted into a game instance immediately before a session. Several files have older modification dates but recent filesystem creation context. That pattern may reflect preserved archive metadata rather than clock manipulation. Compare the archive, extraction behavior, and journal records before describing a cause.

Report checklist

Volume and record identity are retained.

Sequence or reuse context is considered.

Timestamp fields are not merged into one date.

Filesystem changes and execution are reported separately.

Sources & further reading

Microsoft: Master File Table ↗

MFTECmd: author documentation ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Recycle Bin & deletion records

Examine discarded files and deletion metadata while accounting for pathways that bypass the Recycle Bin.

Question & scope

The Recycle Bin can preserve content and metadata for files deleted through supported shell operations. During a screenshare examination, it may reveal a removed mod, archive, or utility connected to the disputed session. Examine the relevant volume and user context rather than only the graphical Recycle Bin view. The GUI is a presentation of available items, not a complete audit of every deletion.

On modern Windows systems, paired recycle artifacts can contain metadata and the discarded object, commonly using \$I and \$R naming. Preserve the pair and original source location when possible. Use a version-aware parser instead of inferring every field from filenames.

What to collect

Recycle artifacts: Relevant \$Recycle.Bin content from the examined volume and user context.

Metadata pair: Original path, size, deletion-related time, and parser-supported identifiers.

Discarded content: The paired object, its content hash, and acquisition notes.

Corroboration: MFT or USN observations, original mod references, and available execution evidence.

Examination workflow

Record volume and user context before exporting items from the bin.

Collect the artifacts without restoring items into their original paths. Restoration would change the examined system.

Parse metadata with a documented tool such as RBCmd and retain the raw pair.

Hash recovered content and compare it with the exact claimed release before examining its code or resources.

Connect deletion-related records to the session timeline, checking timezone and corroborating filesystem evidence.

Interpretation & false positives

Some deletion mechanisms bypass the Recycle Bin. An empty bin does not prove that no files were deleted, and surviving items may relate to an older or unrelated session. The metadata concerns the discarded object; it does not establish that the program ran before deletion.

A recovered item can still be incomplete, altered, or misidentified. Verify the bytes you have rather than assuming that a displayed original filename authenticates the contents. Do not execute recovered binaries or archives merely to see what they do.

Worked scenario

A discarded JAR matches an unofficial modification, but there is no link to the active game instance. Report its presence and deletion metadata, then look for launcher and game-session context. Possession, installation, loading, and actual use are different questions and should not be merged.

Report checklist

The relevant volume and user context are known.

Paired metadata and content are retained.

Recovery did not silently restore files on the source system.

Deletion is not treated as proof of prior execution.

Sources & further reading

RBCmd: Recycle Bin parser ↗

Microsoft: deleted MFT record reuse ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.

Browser downloads & profile context

Connect downloaded mods and utilities to recorded URLs, target paths, profiles, and exact file identity.

Question & scope

Browser artifacts can help investigate how a mod, archive, or utility reached a computer. Begin by identifying the actual browser profile relevant to the examination. A default profile path is a starting point, not proof that the displayed session used that profile. Chromium-based products, Firefox, portable builds, and alternate profiles use different layouts.

A download entry may preserve a URL chain, target path, and state information. Interpret those fields according to the browser version and database schema. Opening the browser or clearing data during a live check can change the source, so collect relevant copies before using a viewer when possible.

What to collect

Browser profile: Product, version, profile directory, and how it was identified.

Database context: Relevant history or download database and associated journals or WAL files.

Download record: Recorded URLs, target path, timestamps, size or state fields where supported.

Current object: Recovered or surviving file, content hash, and exact release comparison.

Examination workflow

Determine the active profile through the browser's own profile or version information and document the selected path.

Acquire relevant database files coherently. When SQLite WAL data is present, preserve the supporting files rather than copying an arbitrary main file alone.

Use a schema-aware viewer or parser and retain the raw database record identifiers.

Follow recorded redirect or download chains and compare the target path with filesystem records.

Hash surviving downloaded content and assess the exact file; do not assume the URL or download filename proves its bytes.

Interpretation & false positives

A download record is not proof of execution or game use. An incomplete download, a changed target path, an overwritten file, or a later replacement can separate the recorded URL from the bytes currently on disk. Browser history can also reflect redirects, embedded requests, or another user context.

Private browsing, profile differences, cleanup, synchronization, and retention affect coverage. Missing history is not evidence that no download occurred. Public findings should avoid publishing unrelated browsing history or URLs containing account tokens.

Worked scenario

A recorded download points to a legitimate mod release, but the current JAR hash differs. Preserve both observations. Investigate whether the file was later modified, replaced, or copied from another profile. The legitimate URL does not authenticate the current bytes; the mismatch does not explain itself.

Report checklist

The selected browser profile is identified.

Supporting database files are collected coherently.

Download, file identity, and execution claims are distinct.

Private URLs and unrelated browsing data are redacted.

Sources & further reading

Chromium: user data directories ↗

SQLite: write-ahead logging ↗

The workflow and scenario are authored investigation guidance. Confirm source-system and parser support for the build you examine.

Complete reference and source links

Includes the full teaching text, collection checklist and worked scenario from this field reference.