Triager: An Open-Source Platform for Automated Forensic Triage and Investigation

Cristian H. M. Souza[1,2] **, Eduardo O. Chavarro**[1] **, Daniel M. Batista**[2]

[1]Global Emergency Response Team (GERT), Kaspersky
[2]Department of Computer Science, University of São Paulo

{cristian.souza, eduardo.ovalle}@kaspersky.com, batista@ime.usp.br

Abstract. *Digital Forensics and Incident Response (DFIR) investigations commonly require analysts to process heterogeneous Windows artifacts, execute multiple specialized parsers, normalize their outputs, and correlate evidence across several endpoints under strict time constraints. This paper presents* `Triager`*, an open-source DFIR automation and investigation platform for Windows triage collections.* `Triager` *orchestrates evidence extraction, performs artifact analysis, normalizes results into CSV files, and organizes the findings in a consistent investigation-ready structure. The Web Console imports these results into per-case databases and provides multi-case and multi-machine management, indexed search, filtering, cross-host correlation, unified timelines, IOC scanning, AI-assisted analysis, evidence export, and report generation. By integrating evidence processing and investigation management,* `Triager` *reduces repetitive operational work while preserving the use of specialized forensic parsers for artifact extraction.*

1. Introduction

Digital Forensics and Incident Response (DFIR) investigations depend on the correct collection, parsing, and analysis of several operating system artifacts. In Windows environments, artifacts such as Event Logs, Prefetch, Amcache, Shimcache, SRUM, Scheduled Tasks, WMI, Registry hives, MFT, USN Journal, browser history, Jump Lists, and PowerShell history may provide important evidence about execution, persistence, user activity, lateral movement, and file system changes [Easttom et al. 2024, Javed et al. 2022]. However, parsing these artifacts manually is time-consuming. Each artifact usually requires a specific parser, generates a specific output format, and needs to be interpreted in the context of the investigation.

During incident response, this creates operational overhead, especially when analysts need to work with several triage packages collected from different machines [Nelson et al. 2025]. This work introduces `Triager`, a Python-based DFIR automation platform designed to automatically process Windows forensic artifacts collected during incident response and post-incident investigations. Instead of manually running dozens of tools and organizing heterogeneous outputs, `Triager` orchestrates parsing and analysis of forensic artifacts, stores their results in a consistent directory layout, and enables post-processing activities such as text search, Indicator of Compromise (IOC) hunting, and Artificial Intelligence (AI)-assisted report generation. The tool is designed to work against triage collections, for example, packages collected using KAPE[1] or Ve-

[1]https://www.sans.org/tools/kape

lociraptor[2].

The remainder of this paper is structured as follows: Section 2 presents background on the Windows forensic artifacts covered by `Triager`, as well as related tools. Section 3 describes the architecture of the proposed solution in detail. Section 4 provides guidance on installation and usage, detailing the various available options. Section 5 highlights the planned demonstration during the conference. Finally, Section 6 concludes the paper with a summary of the tool's capabilities and its potential impact on the DFIR community.

2. Background and Related Tools

Windows systems store several forensic artifacts that can be used to reconstruct user and system activity. Some artifacts provide evidence of program execution, while others help analysts identify persistence mechanisms, file system changes, network usage, or user interaction. Examples of relevant artifacts include:

- **Event Logs (EVTX):** contain records generated by Windows and installed applications. They are useful for reconstructing authentication attempts, account activity, service installation and execution, process creation, PowerShell activity, security alerts, remote access, system errors, and configuration changes [Połczyński 2023]. Correlating events across multiple logs can help establish an incident timeline and identify suspicious behavior.
- **Prefetch:** provides reliable evidence that an application was executed on the system. Prefetch files may contain the executable name, execution count, last execution timestamps, referenced files and directories, and volume information [Neyaz and Shashidhar 2022]. These artifacts are especially useful for confirming the execution of malware, administrative utilities, and attacker tools.
- **Amcache:** stores metadata about applications and binaries observed by Windows, including file paths, names, sizes, hashes, compilation-related information, and publisher details [Souza 2025]. It can help identify suspicious executables, previously present malware, portable tools, and binaries that may no longer exist on disk [Souza et al. 2026].
- **Shimcache:** records historical application compatibility information maintained by Windows. Depending on the operating system version, it may provide file paths, modification timestamps, and execution-related indicators [Joo et al. 2023]. Shimcache is particularly valuable when Prefetch is disabled, unavailable, or has already been deleted, although its entries should not always be interpreted as definitive proof of execution.
- **BAM/DAM:** respectively, Background Activity Moderator and Desktop Activity Moderator, contain execution-related artifacts associated with individual user accounts [Javed et al. 2022]. These registry entries may identify executable paths and timestamps, helping analysts determine which user may have launched a particular program and when the activity occurred.

[2]https://github.com/Velocidex/velociraptor

- **SRUM:** stores detailed information about application usage, network activity, and resource consumption. It may reveal which applications communicated over the network, the user associated with the activity, transferred data volumes, interface usage, and execution periods [Joo et al. 2023]. This information can support the identification of data exfiltration, command-and-control communication, and unusual application behavior.
- **Scheduled Tasks and WMI:** are commonly used by Windows and legitimate software for automation and system administration. However, attackers frequently abuse scheduled tasks, Windows Management Instrumentation (WMI) event subscriptions, consumers, and filters to execute commands, maintain persistence, or trigger malicious activity at specific times or system events [Dieterich et al. 2023]. Their configuration should therefore be reviewed for unusual commands, paths, users, and triggers.
- **Windows Defender Logs:** may contain malware detection history, antivirus verdicts, affected file paths, threat names, remediation actions, exclusion changes, scan results, and protection-related events [Sharikov and Mokrinskii 2025]. These records can reveal previously detected or quarantined malware, even when the original malicious file is no longer available.
- **MFT, USN Journal, and $LogFile:** provide complementary information about file-system activity on NTFS volumes. Together, they can help reconstruct file creation, deletion, renaming, and modification events [Sondarva et al. 2023]. They are useful for identifying dropped malware, deleted tools, ransomware activity, staging directories, and attempts to remove evidence [Souza et al. 2025].
- **User artifacts:** include Jump Lists, Recent Files, MRUs, Typed Paths, browser history, PowerShell history, thumbnails, RDP cache, shellbags, and other user-specific records [Chand et al. 2025]. These artifacts can reveal files accessed by users, executed commands, visited websites, navigated directories, remote systems, opened documents, and other interactive activity relevant to an investigation.

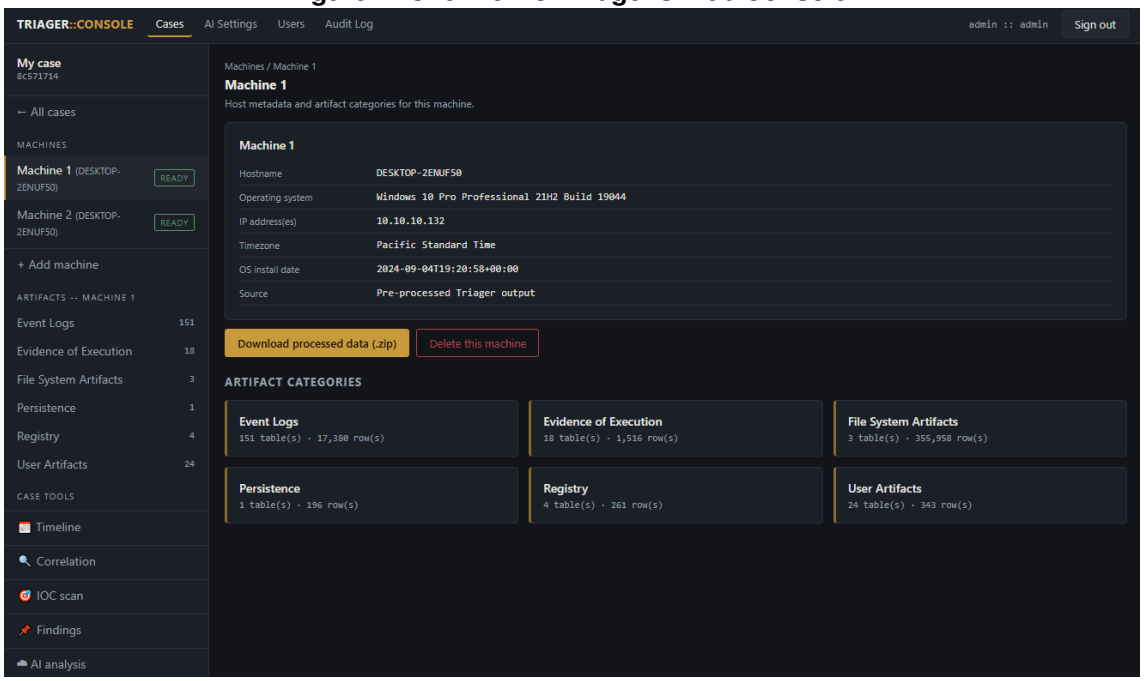
Several specialized tools already exist for parsing Windows forensic artifacts, including `PECmd`, `MFTECmd`, `AppCompatCacheParser`, `SBECmd`, `JLECmd`, `EvtxECmd`, `Hayabusa`, `Chainsaw`, `APT-Hunter`, `SrumECmd`, `AmcacheParser`, and `UserAssistReport`. These utilities are widely used in DFIR investigations and provide mature, high-quality artifact extraction capabilities. `Triager` does not attempt to replace these tools or unnecessarily reimplement their parsers. Instead, it uses them as components of a broader DFIR processing and investigation platform.

At the processing layer, `Triager` identifies the artifacts available in a triage collection, invokes the appropriate forensic utilities, performs additional native extraction where required, normalizes heterogeneous outputs, and organizes the resulting evidence into a consistent investigation-ready structure. Its Web Console (Figure 1) adds a complete investigation layer over the processed evidence, including multi-case and multi-host management, indexed search, filtering, cross-machine correlation, unified timelines, IOC scanning, AI assisted analysis, evidence packaging, and report generation.

This approach preserves the reliability of established forensic parsers while addressing the operational work that occurs around them: configuring tools, processing col-

lections, locating outputs, normalizing data, correlating artifacts, managing cases, documenting findings, and producing investigation reports. The Command Line Interface (CLI) provides the processing and analysis engine, while the Web Console transforms its normalized outputs into a collaborative workspace for examining and correlating evidence across artifacts, machines, and cases. By combining these layers, `Triager` reduces repetitive work and allows investigators to focus on analysis, interpretation, and incident reconstruction.

Figure 1. Overview of Triager's Web Console.



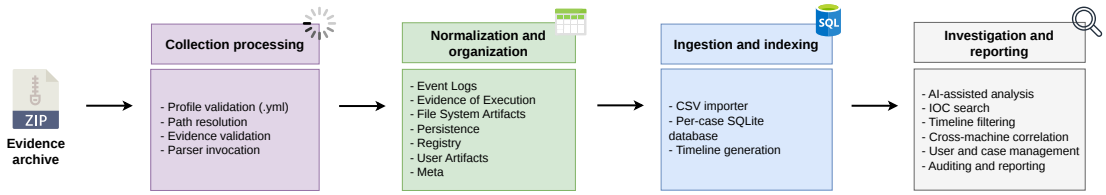
3. Architecture and Implementation

`Triager` is implemented in Python and follows a modular execution model. The tool receives a triage collection as input, resolves the expected artifact paths using a configuration file, creates an output directory, and executes the available parsers in parallel. The architecture, summarized in Figure 2, contains four main layers:

- **Collection processing layer:** the CLI receives a triage directory or ZIP archive, loads a built-in or custom collection profile, resolves artifact paths, validates the available evidence, and invokes the required forensic parsers. Velociraptor is the default profile, while custom YAML profiles can be supplied for other collection layouts.
- **Normalization and organization layer:** parser outputs are stored as CSV files under forensic categories: Event Logs, Evidence of Execution, File System Artifacts, Persistence, Registry, User Artifacts, and Meta. The Meta directory contains the effective configuration and host information extracted from Registry hives, including hostname, operating-system information, IP addresses, timezone, installation date, installed software, and autoruns.

- **Ingestion and indexing layer:** the Web Console supports two ingestion paths. Raw evidence archives are extracted, processed by the same engine as the CLI, and then imported. Previously processed `Triager` output can also be imported directly without rerunning the parsers. Each CSV is loaded into a machine-namespaced table inside a shared per-case SQLite database, so that both single-machine browsing and case-wide correlation are simple queries against one database rather than a federation of separate files.
- **Investigation and reporting layer:** the Web Console exposes the imported evidence through authenticated, role-aware case views. Investigators can browse and filter tables, export CSV data, search across artifacts, correlate values between machines, build a unified timeline, scan IOC lists, create findings linked to artifact-row snapshots, interact with OpenAI-compatible or Claude-compatible AI providers, download processed evidence packages, and generate Word reports.

Figure 2. Triager architecture.

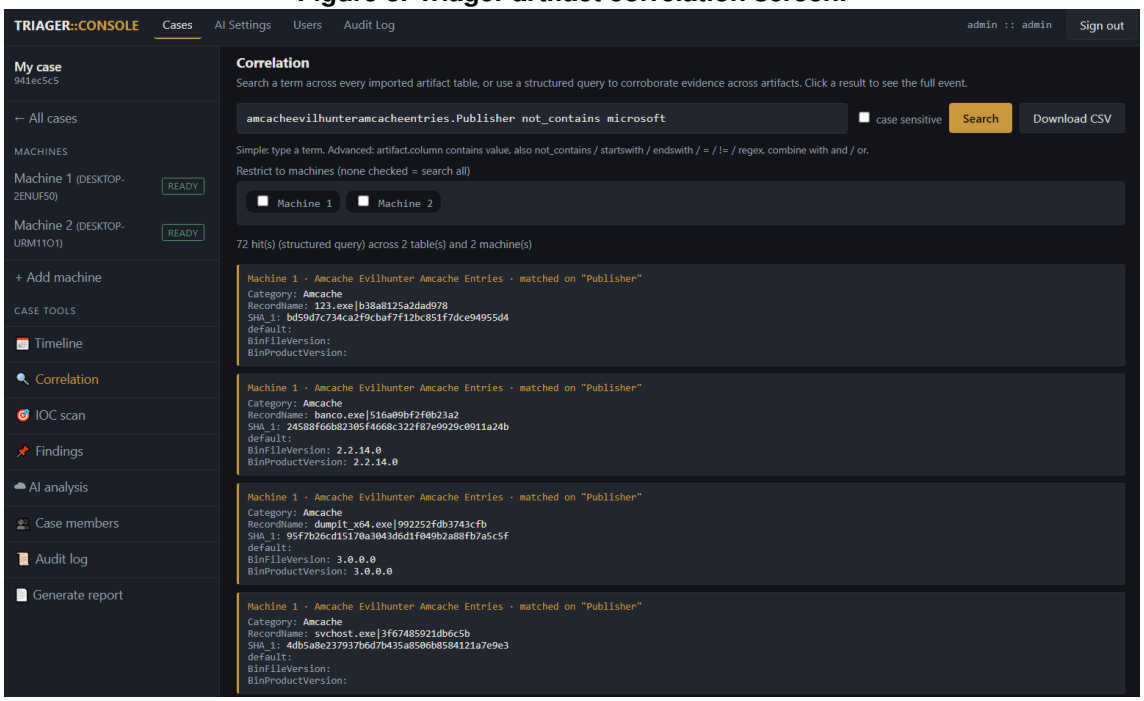


For cross-artifact analysis, the Web Console exposes a structured correlation query language (e.g., `amcache.publisher not_contains microsoft`), supporting per-column filters, AND/OR combination, and an optional case-sensitive mode (Figure 3). Investigators can flag specific artifact rows as findings, which persist a JSON snapshot of the row at flagging time so a finding remains meaningful even if the underlying table is later re-imported; an audit log records every case-relevant action (logins, ingestion, exports, IOC scans, AI questions) for accountability; and role-based access control distinguishes editing investigators from read-only reviewers. Processed evidence, table data, and timeline or correlation results can all be exported as CSV, and a full Word (`.docx`) investigation report can be generated directly from the web interface.

AI support is implemented as an optional investigation aid rather than an autonomous decision mechanism. Investigators select the artifact scope (case-wide, a single machine, or a single artifact table), submit questions, and receive responses within persistent, per-scope conversation threads. Supported integrations include OpenAI-compatible chat-completion endpoints[3] and Anthropic Claude[4], allowing the use of either an external provider or a self-hosted/local model, as shown in Figure 4. The artifact context sent to the model is assembled with a category-aware budget: evidence-of-execution and persistence artifacts receive a larger share of the budget than routine event-log channels, and table columns that are empty across the sampled rows are pruned before serialization. Because evidence may contain confidential information, investigators must use authorized endpoints and follow applicable data-handling requirements.

[3]https://github.com/openai/openai-python
[4]https://platform.claude.com

Figure 3. Triager artifact correlation screen.



`Triager` requires Python 3.10 or newer. Its main dependencies are: `PyYAML`, `python-registry`, `FastAPI`, `SQLAlchemy`, and `Uvicorn`. The external forensic utilities are placed under the `tools/` directory. Examples of supported tools include `PECmd`, `MFTECmd`, `AmcacheParser`, `AppCompatCacheParser`, `SBECmd`, `JLECmd`, `LECmd`, `EvtxECmd`, `SrumECmd`, `RBCmd`, `WxTCmd`, `Hayabusa`, `Chainsaw`, and `APT-Hunter`.

4. Installation and Usage

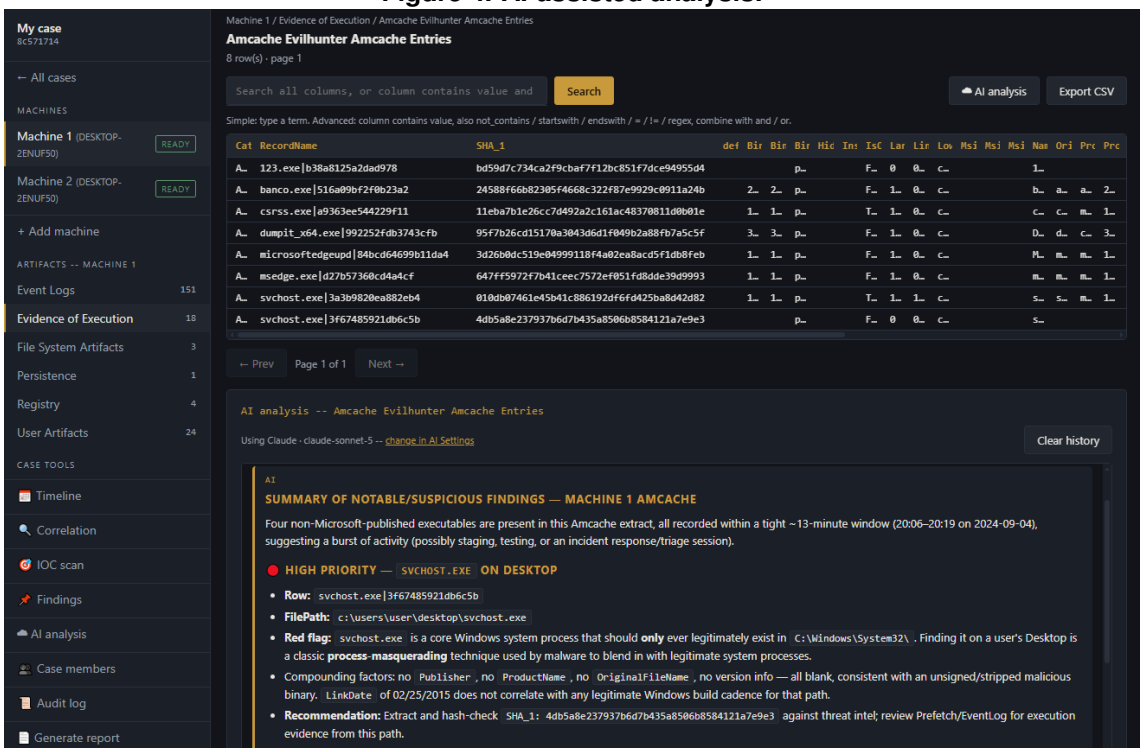
`Triager` is available as an open-source project on `https://github.com/cristianzsh/triager`. A standalone Windows executable can be built with the included `build.bat` script, which packages the CLI, the Web Console, and every third-party parser into a single `Triager.exe` binary. A configuration file can be defined to specify the triage acquisition structure (the default built-in option is Velociraptor). A basic execution against a ZIP collection can be performed with:

```
Triager.exe --zip evidences_machine01.zip ^
--output processed_artifacts
```

Inside the output directory, the following directory tree will be generated by the processing engine:

- Event logs: processed Windows Event Logs.
- Evidence of execution: processed artifacts related to software execution on the analyzed system (e.g., Prefetch, Amcache, SRUM, and antivirus detections).
- File system artifacts: processed Windows NTFS artifacts, such as the MFT, USN Journal, and $LogFile.

Figure 4. AI-assisted analysis.



- Persistence: processed artifacts related to mechanisms that may provide persistence on the system, such as WMI and scheduled tasks.
- Registry: parsed relevant Registry keys, such as BAM, DAM, and Shimcache.
- User artifacts: processed user-related artifacts, such as browsing history, Certutil artifacts, Jump Lists, MUICache, RDP cache, PSReadLine, Shellbags, Thumbnails, UserAssist, and Timeline.
- Meta: information about the host from which the evidence was collected (e.g., operating system, time zone, installation date, computer name, and IP addresses).

After processing the evidence, the analyst can search across the generated output:

```
Triager.exe -d processed_artifacts --search "PsExec"
Triager.exe -d processed_artifacts --search "Mimikatz" ^
--search-max-hits 50
```

`Triager` also supports IOC hunting using a text file containing one indicator per line:

```
Triager.exe -d processed_artifacts --find-iocs iocs.txt ^
--save-iocs iocs_dir
```

Finally, the Web Console can be started from the very same binary (with the `--web` option or by just double-clicking it). Correlation, timeline construction, IOC scanning, findings, AI-assisted analysis, and report generation are all performed through this console once a case's evidence has been ingested; the console accepts either raw evidence archives, which it processes with the same engine as the CLI, or already-processed `Triager` output.

5. Demonstration

During the conference, `Triager` will be demonstrated using Windows triage collections containing multiple forensic artifacts. The demonstration will show how an analyst can automatically parse the available evidence and then use the Web Console to investigate the resulting case. The demonstration will include the following steps:

1. Execution of `Triager` against a collected Windows triage package;
2. Automatic parsing of Event Logs, evidence of execution artifacts, persistence artifacts, registry hives, file system artifacts, and user activity artifacts;
3. Review of the generated output directory;
4. Ingestion of the processed evidence into the Web Console, including a second machine's collection to illustrate multi-host case management;
5. Search for suspicious strings, commands, binaries, paths, and IOC hunting using a predefined list;
6. Construction of a unified timeline across both machines, and a structured correlation query spanning artifacts and hosts;
7. AI-assisted analysis of the ingested evidence using a configured provider, and export of the timeline, correlation results, and processed evidence as CSV;
8. Generation of a Word investigation report summarizing the case.

A complete video demonstration about the tool's usage is available at `https://www.youtube.com/watch?v=sqB2SV0xDWg`.

6. Conclusion

This paper presented `Triager`, an open-source DFIR automation and investigation platform for Windows triage collections. Its processing engine orchestrates forensic parsers, extracts host data from registry hives, and normalizes heterogeneous outputs into a consistent, investigation-ready structure, while also supporting command-line search and IOC hunting directly against processed evidence. Its Web Console extends this engine into a collaborative investigation workspace, adding multi-case and multi-machine management, indexed and structured cross-machine correlation, a unified timeline, IOC scanning, AI-assisted analysis with pluggable providers, and automated report generation, all distributed as a single binary.

As future work, we plan to broaden artifact and collection-profile coverage, add timestomping-detection heuristics to the timeline, extend correlation across cases, and give investigators finer-grained control over what evidence is exposed to AI providers, particularly for air-gapped or highly sensitive engagements where only local models should be used.

References

Chand, R. R., Sharma, N. A., and Kabir, M. A. (2025). Advancing web browser forensics: Critical evaluation of emerging tools and techniques. *SN Computer Science*, 6(4):355.

Dieterich, A., Schopp, M., Stiemert, L., Steininger, C., and Pöhn, D. (2023). Evaluation of persistence methods used by malware on microsoft windows systems. In *ICISSP*, pages 552–559.

Easttom, C., Butler, W., Phelan, J., Bhagavatula, R. S., Steuber, S., Rodriguez, K., Balkissoon, V. I., and Naseer, Z. (2024). *Windows Forensics*. Springer.

Javed, A. R., Ahmed, W., Alazab, M., Jalil, Z., Kifayat, K., and Gadekallu, T. R. (2022). A comprehensive survey on computer forensics: State-of-the-art, tools, techniques, challenges, and future directions. *IEEE Access*, 10:11065–11089.

Joo, D., Lee, J., and Jeong, D. (2023). A reference database of windows artifacts for file-wiping tool execution analysis. *Journal of forensic sciences*, 68(3):856–870.

Nelson, A., Rekhi, S., Souppaya, M., and Scarfone, K. (2025). Incident response recommendations and considerations for cybersecurity risk management. *NIST special publication*.

Neyaz, A. and Shashidhar, N. (2022). Windows prefetch forensics. In *Breakthroughs in Digital Biometrics and Forensics*, pages 191–210. Springer.

Połczyński, P. (2023). Analysis of event logs in computers (windows systems).

Sharikov, P. and Mokrinskii, N. (2025). Analyze windows logs to investigate phishing attack. In *2025 International Ural Conference on Electrical Power Engineering (UralCon)*, pages 668–672. IEEE.

Sondarva, K. S., Kumar, A., Gohil, B. N., Patel, S. J., Rajvansh, S., and Shah, R. T. (2023). Forensics analysis of ntfs file systems. In *Advances in Cyberology and the Advent of the Next-Gen Information Revolution*, pages 138–165. IGI Global Scientific Publishing.

Souza, C. (2025). Forensic journey: hunting evil within amcache. Technical report, Kaspersky Securelist.

Souza, C. H., Chavarro, E. O., and Batista, D. M. (2026). Amcache-evilhunter: Automating evidence of execution extraction from the amcache.hve artifact. In *Anais Estendidos do XXVI Simpósio Brasileiro de Cibersegurança*, Porto Alegre, RS, Brasil. SBC.

Souza, C. H., Pascoal, T., Neto, E. P., Sousa, G. B., SL Filho, F., Batista, D. M., and Silva, F. S. D. (2025). Sdn-based solutions for malware analysis and detection: State-of-the-art, open issues and research challenges. *Journal of Information Security and Applications*, 93:104145.