International Journal of Safety and Security Engineering
Vol. 16, No. 3, March, 2026, pp. 503-508
Journal homepage: http://iieta.org/journals/ijsse

Trust Propagation Failure in the Recursive-Execution Path of Nested Archives

Haewon Byeon

Department of Future Technology, Korea University of Technology and Education (KOREATECH), Cheonan-si 31253, Republic of Korea

Corresponding Author Email: bhwpuma@naver.com

https://doi.org/10.18280/ijsse.160303

Received: 19 January 2026
Revised: 20 March 2026
Accepted: 25 March 2026
Available online: 31 March 2026

Keywords:
alternate data streams, archive software, IAttachmentExecute, Mark-of-the-Web, nested archives, SmartScreen, trust propagation, Windows security

ABSTRACT

Mark-of-the-Web (MoTW) relies on the new technology file system (NTFS) Zone.Identifier alternate data streams (ADS) to label files originating from untrusted zones and to trigger Windows defenses such as SmartScreen. In early 2025, archive utilities including 7-Zip (v24.09) and Alzip (v12.30) released patches to preserve MoTW when extracting nested archives. Experiments were conducted in Windows 11 Enterprise 24H2 (Build 26100.1742) on an NTFS volume using 7-Zip v24.09 and Alzip v12.30. The design compared Group A (explicit extraction to a user-selected location) with Group B (direct activation through the recursive-execution path inside the archive viewer). This paper shows that the fixes do not consistently cover the recursive-execution path inside an archive viewer. Using a depth-3 nested ZIP test case and Sysinternals Procmon call-stack traces, we compared the patched explicit-extraction path against the recursive-execution path that expands payloads into the temporary directory (TEMP). The results showed that the explicit extraction path preserved Zone.Identifier, whereas the recursive-execution path created a temporary executable without an inherited Zone.Identifier. In both utilities, the temporary executable is created without inheriting the parent archive's Zone.Identifier stream, which may weaken SmartScreen and related trust checks. The experimental evidence directly demonstrates provenance loss in the tested workflow; the broader security implication is that downstream trust enforcement can be weakened when origin marking is missing. We interpret the behavior as a path-specific implementation gap consistent with code-path divergence: vendors patched the primary extraction routine but left a legacy temporary-execution routine unchanged. To close the gap, we propose shifting trust marking to the Windows delegation model via IAttachmentExecute, including SetReferrer-based provenance propagation, so that derived files receive OS-consistent policy application regardless of the UI workflow.

1. INTRODUCTION

Windows relies on provenance-related metadata to distinguish files that arrive from untrusted sources such as the web or email. In Windows environments, origin information associated with downloaded content and new technology file system (NTFS) alternate data streams (ADS) remains important for both security enforcement and forensic interpretation [1, 2]. When that provenance is dropped somewhere in the tool chain, downstream trust decisions can become less reliable.

Modern intrusion campaigns increasingly distribute payloads inside archives and use obfuscation, packing, or multi-stage delivery to complicate inspection. Recent work on malware detection, fileless malware, and packing-aware analysis shows that transformed or nested payloads can reduce the effectiveness of purely static inspection and increase analytical complexity [3-6].

Archive viewers add a second problem. They encourage a convenience workflow: browse the archive and double-click an item to run it. Recent usable-security research shows that user decisions are strongly shaped by interface cues, risk indicators, usability preferences, and habituated behavior, which can weaken the protective value of warnings when execution appears routine [7, 8].

In early 2025, major archivers shipped patches to preserve Mark-of-the-Web (MoTW) for nested archives. We examine patched 7-Zip v24.09 and Alzip v12.30 and show that the protection is concentrated in the explicit extraction routine, not the archive-viewer execution routine.

In this paper, recursive-execution path refers specifically to the workflow in which a user opens a nested archive inside the archive viewer and directly activates an inner payload, causing the application to materialize the file into a temporary directory (TEMP) and execute it. This term is used to distinguish that workflow from ordinary recursive extraction, where files are explicitly unpacked to a user-selected destination.

Despite recent advances in malware analysis, packing-aware detection, and multi-stage threat modeling [3-6], it remains unclear whether all file-creation workflows inside archive software preserve provenance consistently. Most prior

work has focused on malicious delivery, detection, and reconstruction at a broader level, but less attention has been given to execution-oriented viewer paths that users can reach through ordinary browsing behavior. This leaves an important research gap: a patch may appear effective on the main extraction routine while still missing a secondary file-creation path.

This study, therefore, asks whether patched archivers preserve MoTW equally across explicit extraction and recursive-execution paths, and whether any residual gap can be explained by path-specific implementation differences. The paper makes three contributions. First, it experimentally verifies the residual gap using a controlled depth-3 nested archive. Second, it distinguishes the observed workflows using process attribution and call-stack evidence. Third, it proposes a more robust OS-delegated mitigation strategy that targets the specific file-creation point identified in the experiments.

2. RELATED WORK

2.1 Mechanisms of Mark-of-the-Web and fragility of new technology file system alternate data streams

Windows commonly records origin-related metadata for external content using mechanisms closely tied to NTFS and browser- or download-generated artifacts. Recent work on ADS and Windows-based browser forensics reinforces that provenance can be technically available yet operationally fragile across file-handling workflows [1, 2].

2.2 Archive-based evasion and limits of deep inspection

Nested containers and transformed payloads continue to stress automated analysis pipelines. Recent surveys and empirical studies on malware detection, fileless malware, and packing-aware analysis show that obfuscation, packing, and execution without stable on-disk signatures complicate both detection and attribution [3-6].

2.3 Human factors and warning bypass

User behavior matters because provenance-based security only has value at the point of decision and execution. Recent studies in usable security and privacy decision-making report that visual risk indicators can improve detection behavior, but that usability preferences, inertia, and habituation still shape whether users heed security-relevant cues [7, 8].

2.4 Forensic artifacts and provenance reconstruction

When an archive viewer extracts and executes a file transiently, investigators still need a defensible account of what ran and where it came from. Recent digital-forensics work on evidence containers, enriched timelines, event reconstruction, and the preservation of meaning in evolving systems highlights the importance of retaining context, reference data, and temporal relationships when reconstructing system behavior [9-12].

2.5 Anti-forensics, evidence containers, and comparison methods

Metadata loss along a processing path can therefore weaken evidentiary interpretation even when the payload content itself is unchanged. Recent systematic work on multi-stage threat behaviors and provenance-graph-based reconstruction likewise underscores the importance of preserving causal context rather than only the final artifact [13, 14]. Foundational usable-security research on user behavior and malware warnings further explains why provenance-based safeguards can fail at the point of execution, even when warnings are technically available [15, 16]. Earlier work on formalizing digital forensic artifacts, bulk triage, and Windows interaction artifacts also remains relevant when analysts must reconstruct transient execution workflows [17-19]. In addition, forensic methods for memory-structure inspection, timestamp provenance analysis, and similarity-based comparison of near-duplicate files reinforce why investigators must track derivation paths rather than only terminal files [20-22]. In this study, the broader digital forensics and threat-reconstruction literature is retained only as conceptual background, while the experimental emphasis is placed on the current trust-propagation problem in patched Windows archivers.

3. THREAT MODEL: CODE PATH DIVERGENCE HYPOTHESIS

3.1 Attack scenario: Depth-3 recursive execution

We model a depth-3 scenario in which an attacker wraps an executable inside multiple nested archives to exploit both scanner depth limits and provenance loss. The user downloads an outer ZIP from the web, which assigns ZoneId = 3. The user then navigates into the inner ZIP within the archiver UI and finally double-clicks the payload executable.

The recent patches cover the first transition: when the archiver materializes the inner ZIP into a temporary location, it propagates Zone.Identifier correctly. The vulnerable step is the second transition: when the user activates the payload inside the inner ZIP, the archiver extracts the executable into a random TEMP subdirectory and launches it.

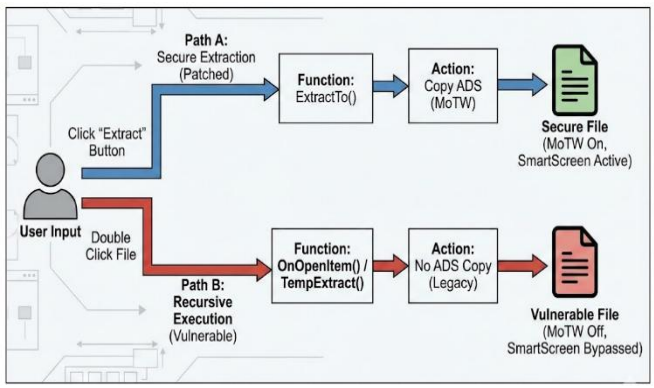

Figure 1. Code path divergence model

This section is an analytical frame rather than a statement of directly observed fact. The experiments below establish differences in file creation, call-stack traces, and Zone.Identifier preservation. The code-path divergence explanation is presented later as an interpretation consistent with those observations. Figure 1 illustrates a model of hypothetical paths divided into explicit extraction and recursive execution.

3.2 Architectural divergence hypothesis

We hypothesize that archivers implement two distinct internal pipelines: a hardened extraction pipeline used for explicit "Extract" actions, and a legacy viewer execution pipeline used for double-click launches. Security patches targeted the primary extraction routine, while the temporary-execution routine remained unchanged.

3.3 Comparative analysis

Table 1 defines the comparison frame between the two workflows and clarifies why the same archive content can produce different trust outcomes depending on user interaction.

Table 1. Technical and security characteristics of explicit extraction and recursive execution paths

Aspect	A (Explicit Extraction) vs. B (Recursive Execution)
User intent	A: Persistent storage and ownership; B: Quick inspection and immediate execution
Destination	A: User-chosen location (e.g., Desktop, Downloads); B: Temporary folder (TEMP\RandomDir\...)
Estimated internal entry point	A: CPanel::ExtractTo(); B: CPanel::OnOpenItem()
Patch coverage	A: Covered (patched); B: Missed (legacy / unpatched)
Alternate data streams (ADS) handling	A: Creates Zone.Identifier after WriteFile; B: Closes handle without creating Zone.Identifier
Security outcome	A: SmartScreen expected to trigger (trust preserved); B: Trust evidence can be missing (trust washed)

3.4 Trust washing in the temporary directory

When the recursive-execution path writes an executable into TEMP without Zone.Identifier, Windows treats the object as a locally created file. The directory itself is not inherently trusted, but the OS loses the "untrusted origin" marker that normally triggers friction at launch. At that point, the payload has been laundered: its content is unchanged, yet its provenance signal is gone.

4. EXPERIMENTS AND RESULTS

4.1 Setup and sample preparation

We ran the experiments on Windows 11 Enterprise 24H2 (Build 26100.1742) on an NTFS volume to ensure full ADS support. We tested 7-Zip v24.09 (x64) and ESTsoft Alzip v12.30. The sample payload was a benign calculator executable (PoC_Calc.exe). The archive structure was Outer.zip (ZoneId = 3) -> Inner.zip -> PoC_Calc.exe. We verified that Outer.zip carried Zone.Identifier before each run.

Before each run, Outer.zip was verified to carry a Zone.Identifier ADS by PowerShell-based inspection of the ADS. The experiment was performed under the standard Windows 11 desktop configuration used for ordinary execution. Procmon collection was configured with filters for Process Name is 7zFM.exe or AlZip.exe, Operation is CreateFile/WriteFile/CloseFile, and Path contains the temporary output location or extracted payload name; call stacks were captured at the relevant WriteFile events.

The experiment used a depth-3 nested ZIP and a single benign executable payload to isolate the provenance-propagation phenomenon under controlled conditions. This design is sufficient to demonstrate that the gap exists, but it does not by itself establish prevalence across other nesting depths, payload classes, archive formats, or vendor implementations.

4.2 Experimental procedure

We varied only the user interaction. Group A (control) navigated to PoC_Calc.exe and used the toolbar Extract action to save it to the Desktop, then launched it. Group B (experimental) double-clicked PoC_Calc.exe inside the archive UI to trigger the direct-execution workflow. We captured file I/O with Sysinternals Procmon, filtered for the archiver process and file operations, and collected call stacks at the WriteFile events. Table 2 summarizes the experimental design in a structured form so that the comparison logic is explicit.

Table 2. Experimental design in comparative logic

Element	Description
Environment	Windows 11 Enterprise 24H2, new technology file system (NTFS), 7-Zip v24.09, Alzip v12.30
Archive structure	Outer.zip (ZoneId = 3) -> Inner.zip -> PoC_Calc.exe
Fixed variables	OS, file system, archive content, payload, tool version, capture method
Independent variable	User workflow: explicit extraction vs. recursive execution inside archive viewer
Group A	Extract PoC_Calc.exe to Desktop, then launch
Group B	Activate PoC_Calc.exe directly inside archive viewer
Observed outcome variables	Process attribution, call-stack path, Zone.Identifier presence/absence, output location

Result 1: Process attribution

Table 3. Procmon event log: File creation attribution in the recursive-execution workflow (Group B)

Event	Evidence and Interpretation
+0.000 s CreateFile	7zFM.exe (PID 4120) → TEMP\TEMP_7zE3\PoC_Calc.exe (SUCCESS); Archiver creates the payload file directly.
+0.002 s WriteFile	7zFM.exe (PID 4120) → TEMP\TEMP_7zE3\PoC_Calc.exe (SUCCESS); Writes payload data (size: 28 KB).
+0.015 s CloseFile	7zFM.exe (PID 4120) → TEMP\TEMP_7zE3\PoC_Calc.exe (SUCCESS); Closes file handle without writing Zone.Identifier.
+0.050 s CreateFile	Explorer.exe (PID 5508) → TEMP\TEMP_7zE3\PoC_Calc.exe (SUCCESS); Shell accesses file for execution.

A first question is attribution: does the shell create the

temporary payload, or does the archiver create it? Table 3 provides direct process-attribution evidence for temporary file creation in the recursive-execution path. The log contains no subsequent Zone.Identifier write between WriteFile and CloseFile, which is where a manual propagation patch would normally appear.

Result 2: Divergent code paths verified by call stacks

Table 4. Call stack comparison at WriteFile between explicit extraction (Group A) and recursive execution (Group B) in 7-Zip v24.09

Stack Frame	A (ExtractTo) vs. B (OnOpenItem)
Frame 00	A: Kernel32.dll!WriteFile+0x14; B: Kernel32.dll!WriteFile+0x14
Frame 01	A: 7z.dll!NWindows::NFile::NIO::COutFile::Write+0x22; B: 7z.dll!NWindows::NFile::NIO::COutFile::Write+0x22
…	A: …; B: …
Frame 05	A: 7z.dll!CArchiveExtractCallback::GetStream; B: 7z.dll!CArchiveExtractCallback::GetStream
Frame 08	A: 7zFM.exe!CPanel::ExtractTo (offset: +0x110); B: 7zFM.exe!CPanel::OnOpenItem (offset: +0x8C)
Frame 09	A: 7zFM.exe!CApp::OnCommand (CmdID: Extract); B: 7zFM.exe!CApp::OnCommand (CmdID: Open)
Post-action	A: Calls SetFileInformationByHandle (patch path); B: No security API call observed

Call-stack traces show where the behavior diverges. In Group A, the stack passes through CPanel::ExtractTo, which is the module vendors patched. In Group B, the stack passes through CPanel::OnOpenItem, which drives temporary extraction for viewing or execution. Table 4 identifies the path split at the call-stack level.

Result 3: Forensic verification of the final artifact

We then verified the final on-disk state using PowerShell before execution to avoid a time-of-check/time-of-use race. Group A's extracted file retained a Zone.Identifier stream, so SmartScreen and related checks could activate. Group B's temporary file had the same content hash yet lacked the stream, meaning the trust evidence was missing. Figure 2 shows a representative output.

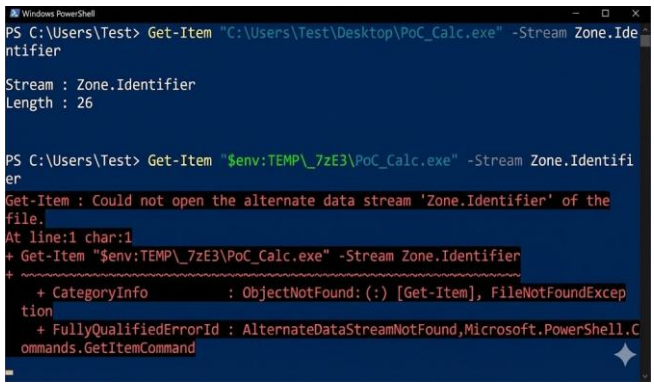

Figure 2. PowerShell verification output for Zone.Identifier presence

Alzip v12.30 exhibited the same pattern: the explicit extraction routine preserved MoTW for derived files, while the double-click execution routine produced a temporary executable without inherited Zone.Identifier. Table 5 summarizes the end-to-end findings and links each result to its evidence source.

Table 5. The end-to-end findings

Tool	Workflow	Output Location	Zone.Identifier Status	Evidence Source	Security Implication
7-Zip v24.09	Explicit extraction	User-selected path	Preserved	PowerShell + call stack	Trust preserved
7-Zip v24.09	Recursive execution	%TEMP%	Missing	Procmon + PowerShell	Provenance gap
Alzip v12.30	Explicit extraction	User-selected path	Preserved	PowerShell	Trust preserved
Alzip v12.30	Recursive execution	%TEMP%	Missing	PowerShell + workflow comparison	Provenance gap

5. DISCUSSION

5.1 Structural limits of manual propagation

Most vendor patches implement MoTW propagation as a post-write copy: after writing the primary stream, the tool creates Zone.Identifier and copies the parent's metadata. This approach is brittle. Archivers create files through many entry points (toolbar extraction, drag-and-drop, context menu, CLI, viewer execution), and it is easy to miss a path. Even when developers remember the patch site, stream writes can fail independently of the primary file write, leaving a "clean" executable behind with no obvious error.

The experimental evidence does not prove that every archive workflow is vulnerable. Rather, it shows that the specific recursive-execution path observed in this study bypasses the patched explicit-extraction routine. For that reason, the architectural proposal below is presented not as a general redesign mandate, but as a targeted response to the file-creation path identified in the experiments.

5.2 Proposed delegation model via IAttachmentExecute

We propose moving from scattered ADS manipulation to the OS delegation model provided by IAttachmentExecute. The application should report provenance and allow Windows attachment services to apply policy and attach the appropriate trust evidence. This design centralizes decision making, reduces path-specific patchwork, and aligns with recent evidence-handling work that emphasizes preserving context and reference data across transformations [9, 12].

Because the observed gap occurs after temporary-file creation in the recursive-execution routine, the mitigation must target that same point in the workflow. In this sense, IAttachmentExecute::Save() is proposed as a direct remedy for the missing post-write trust-marking step identified in the

experiments, not as a solution disconnected from the evidence.

5.3 Trust inheritance via SetReferrer

SetReferrer matters in nested archives. When the payload is derived from Inner.zip that already carries ZoneId = 3, the tool can call SetReferrer(path_to_inner_zip) and set an appropriate source (e.g., about:internet or the original URL). Windows can then apply a consistent zone decision to the derived executable even though it resides in a local TEMP (Figure 3).

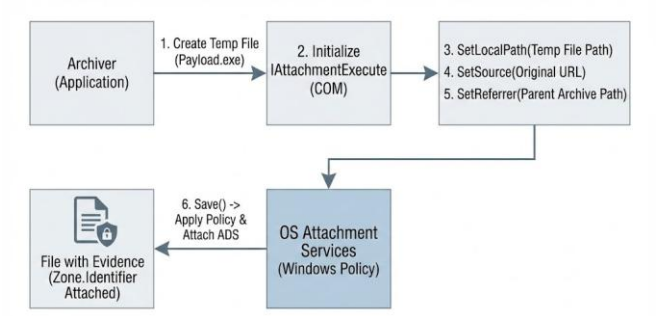

Figure 3. IAttachmentExecute delegation workflow for derived files

5.4 Implementation strategy: Save() vs. Execute()

IAttachmentExecute exposes two policy entry points: Save() and Execute(). For archive viewers, Save() is a better fit. Execute() can launch the file but may interfere with the archiver's UX and error handling (password prompts, progress UI, and custom handlers). Table 6 evaluates implementation options for repairing the specific gap observed earlier.

Table 6. Comparison of IAttachmentExecute methods for policy application in the recursive-execution workflow

Criterion	Save() vs. Execute()
Primary function	Save(): Applies policy and attaches evidence (no execution); Execute(): Applies policy and launches the file
UI control	Save(): Application retains control; can keep existing UX flow; Execute(): Control shifts to the OS; prompts may block UX
Sync / async	Save(): Synchronous; Execute(): Supports asynchronous use (requires message pumping)
Implementation effort	Save(): Lower (Save() then existing ShellExecute/CreateProcess); Execute(): Higher (replaces existing execution logic)
Suitability	Save(): High; Execute(): Medium

5.5 Recommended implementation logic

A practical implementation integrates policy delegation into the viewer-execution routine (e.g., OnOpenItem): create the temporary file, delegate policy with IAttachmentExecute::Save, then launch using the existing execution pathway. The following C++-style pseudocode illustrates the flow.

```
// 1. File generation (existing logic)
std::wstring tempPath = CreateTempFile(payloadData);
// 2. Policy delegation (new security logic)
IAttachmentExecute* pAE = nullptr;
HRESULT hr =
CoCreateInstance(CLSID_AttachmentServices, nullptr,
CLSCTX_INPROC_SERVER,
                 IID_IAttachmentExecute,
(void**)&pAE);
if (SUCCEEDED(hr)) {
  pAE->SetLocalPath(tempPath.c_str());
  pAE->SetSource(L"about:internet"); // or the original URL
  pAE->SetReferrer(parentArchivePath.c_str()); // e.g.,
Inner.zip
  pAE->Save();
  pAE->Release();
}

// 3. Execution (existing logic)
ShellExecute(nullptr, L"open", tempPath.c_str(), nullptr,
nullptr, SW_SHOWNORMAL);
```

5.6 Practical relevance for engineering and security teams

In real-world software use, many users do not extract nested content manually. They browse the archive, inspect inner objects, and launch files directly from the viewer UI. For engineering teams, the finding matters because patch verification limited to the primary extraction function may leave secondary execution-oriented paths untested, even though those paths are reachable through ordinary user behavior. For security teams, the practical consequence is not that a universal SmartScreen bypass has been proven, but that provenance-based friction can be reduced when origin marking is silently lost during transient file creation.

5.7 Limitations and trade-offs

Policy delegation introduces overhead. Creating COM objects and calling Save() for every previewed file can add latency when users browse archives containing thousands of small objects. Tools can mitigate this by applying delegation selectively to executable or script-capable extensions (.exe, .msi, .js, .vbs, .doc*).

A second concern is provenance spoofing: a malicious tool could set a benign source string. Attachment services are not a full trust oracle; Windows still applies additional checks through SmartScreen and antivirus engines. The delegation model is meant to reduce accidental provenance loss, not to certify a tool as trustworthy.

This study also has boundary conditions that should be stated explicitly. The evidence is limited to patched 7-Zip v24.09 and Alzip v12.30, a depth-3 ZIP nesting structure, and a single benign payload under one Windows 11 desktop configuration. These constraints are appropriate for a controlled proof-of-phenomenon experiment, but they do not establish how frequently the same gap appears across other archive formats, greater nesting depths, alternate payload types, or different vendor implementations.

6. CONCLUSIONS

We demonstrated that patched 7-Zip v24.09 and Alzip v12.30 still lose MoTW in the recursive-execution path for depth-3 nested archives. Procmon attribution logs show that the archiver creates the temporary payload, and call stacks show that the recursive-execution path bypasses the patched extraction routine. These findings suggest a structural patch-

coverage gap in the tested workflow rather than supporting a claim about every archive implementation or every execution scenario. Archivers should adopt OS-level policy delegation (IAttachmentExecute::Save with SetReferrer) so that every file-creation path receives consistent provenance marking and policy enforcement.

ACKNOWLEDGMENT

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education (NRF-RS-2023-00237287).

REFERENCES

[1] Bhardwaj, A., Kaushik, K., Maashi, M.S., Aljebreen, M., Bharany, S. (2022). Alternate data stream attack framework to perform stealth attacks on active directory hosts. Sustainability, 14(19): 12288. https://doi.org/10.3390/su141912288

[2] Raza, A., Hussain, M., Tahir, H., Zeeshan, M., Raja, M.A., Jung, K.H. (2024). Forensic analysis of web browsers lifecycle: A case study. Journal of Information Security and Applications, 85: 103839. https://doi.org/10.1016/j.jisa.2024.103839

[3] Bensaoud, A., Kalita, J., Bensaoud, M. (2024). A survey of malware detection using deep learning. Machine Learning with Applications, 16: 100546. https://doi.org/10.1016/j.mlwa.2024.100546

[4] Azeem, M., Khan, D., Iftikhar, S., Bawazeer, S., Alzahrani, M. (2024). Analyzing and comparing the effectiveness of malware detection: A study of machine learning approaches. Heliyon, 10(1): e23574. https://doi.org/10.1016/j.heliyon.2023.e23574

[5] Kara, I. (2023). Fileless malware threats: Recent advances, analysis approach through memory forensics and research challenges. Expert Systems with Applications, 214: 119133. https://doi.org/10.1016/j.eswa.2022.119133

[6] Gibert, D., Totosis, N., Patsakis, C., Le, Q., Zizzo, G. (2025). Assessing the impact of packing on static machine learning-based malware detection and classification systems. Computers & Security, 156: 104495. https://doi.org/10.1016/j.cose.2025.104495

[7] Baltuttis, D., Teubner, T. (2024). Effects of visual risk indicators on phishing detection behavior: An eye-tracking experiment. Computers & Security, 144: 103940. https://doi.org/10.1016/j.cose.2024.103940

[8] Morkonda, S.G., Chiasson, S., van Oorschot, P.C. (2024). Influences of displaying permission-related information on web single sign-on login decisions. Computers & Security, 139: 103666. https://doi.org/10.1016/j.cose.2023.103666

[9] Han, J., Han, M.L., Lee, S., Park, J. (2024). ECo-Bag: An elastic container based on Merkle tree as a universal digital evidence bag. Forensic Science International: Digital Investigation, 49: 301725. https://doi.org/10.1016/j.fsidi.2024.301725

[10] Dreier, L.M., Vanini, C., Hargreaves, C.J., Breitinger, F., Freiling, F. (2024). Beyond timestamps: Integrating implicit timing information into digital forensic timelines. Forensic Science International: Digital Investigation, 49: 301755. https://doi.org/10.1016/j.fsidi.2024.301755

[11] Breitinger, F., Studiawan, H., Hargreaves, C. (2025). SoK: Timeline based event reconstruction for digital forensics: Terminology, methodology, and current challenges. Forensic Science International: Digital Investigation, 53: 301932. https://doi.org/10.1016/j.fsidi.2025.301932

[12] Spichiger, H., Adelstein, F. (2025). Preserving meaning of evidence from evolving systems. Forensic Science International: Digital Investigation, 52: 301867. https://doi.org/10.1016/j.fsidi.2025.301867

[13] Che Mat, N.I., Jamil, N., Yusoff, Y., Mat Kiah, M.L. (2024). A systematic literature review on advanced persistent threat behaviors and its detection strategy. Journal of Cybersecurity, 10(1): tyad023. https://doi.org/10.1093/cybsec/tyad023

[14] Cui, M., Jiang, Z., Li, S., Ma, C., Zhang, K., Yang, P., Feng, H. (2026). MGDA: A provenance graph-based framework for threat detection and attack scenario reconstruction. Computer Networks, 274: 111806. https://doi.org/10.1016/j.comnet.2025.111806

[15] Adams, A., Sasse, M.A. (1999). Users are not the enemy. Communications of the ACM, 42(12): 40-46. https://doi.org/10.1145/322796.322806

[16] Modic, D., Anderson, R. (2014). Reading this may harm your computer: The psychology of malware warnings. Computers in Human Behavior, 41: 71-79. https://doi.org/10.1016/j.chb.2014.09.014

[17] Harichandran, V.S., Walnycky, D., Baggili, I., Breitinger, F. (2016). CuFA: A more formal definition for digital forensic artifacts. Digital Investigation, 18(Suppl. 1): S125-S137. https://doi.org/10.1016/j.diin.2016.04.005

[18] Garfinkel, S.L. (2013). Digital media triage with bulk data analysis and bulk_extractor. Computers & Security, 32: 56-72. https://doi.org/10.1016/j.cose.2012.09.011

[19] Singh, B., Singh, U. (2016). A forensic insight into Windows 10 Jump Lists. Digital Investigation, 17: 1-13. https://doi.org/10.1016/j.diin.2016.02.001

[20] Schuster, A. (2006). Searching for processes and threads in Microsoft Windows memory dumps. Digital Investigation, 3(Suppl. 1): S10-S16. https://doi.org/10.1016/j.diin.2006.06.010

[21] Schatz, B., Mohay, G., Clark, A. (2006). A correlation method for establishing provenance of timestamps in digital evidence. Digital Investigation, 3(Suppl. 1): S98-S107. https://doi.org/10.1016/j.diin.2006.06.009

[22] Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. Digital Investigation, 3(Suppl. 1): S91-S97. https://doi.org/10.1016/j.diin.2006.06.015