With or Without Logs: Memory Forensic Reconstruction of Model Context Protocol (MCP) Activity in Agentic LLM Systems

Abdus Satter[a,b], Makei Salmon[a,b], Lara Muhanna[a,b], Trevor T Spinosa[a,b], Taha Gharaibeh[a,b], Ibrahim Baggili[a,b]

[a] *Baggili(i) Truth (BiT) Lab, Center of Computation & Technology, Baton Rouge, LA, USA*
[b] *Division of Computer Science & Engineering, Louisiana State University, Baton Rouge, LA, USA*

Abstract

Large Language Model agents increasingly rely on the Model Context Protocol (MCP) to invoke external tools for code execution, data access, and API interaction, expanding both automation capabilities and the forensic attack surface of modern AI-enabled systems. MCP interactions often occur within ephemeral runtimes where durable logs and network traces may be incomplete, unavailable, or deliberately removed, limiting post-incident visibility into agent behavior. In this paper, we present MCPRecon, a protocol-aware memory forensics technique for reconstructing MCP interaction traces from volatile memory. MCPRecon leverages the standardized JSON-RPC 2.0 message structure and MCP method semantics to identify, validate, and correlate MCP protocol artifacts directly from RAM, without relying on disk logs, network captures, or transport-specific instrumentation. Given a post-incident memory snapshot, MCPRecon recovers tool discovery results, invocation arguments, execution outcomes, and tool identifiers, and reconstructs request–response traces using JSON-RPC identifiers. We evaluate MCPRecon across multiple real-world MCP clients, deployment modes (stdio and HTTP), and scenarios, demonstrating that forensically relevant MCP artifacts persist in volatile memory and can be reliably reconstructed. Finally, we illustrate the forensic relevance of recovered artifacts through a successful proof-of-concept attack involving malicious MCP tool usage in Composer 1 and Gemini 3 Flash, showing how memory-resident protocol evidence enables attribution and post-incident analysis when traditional logs are absent.

Keywords: Memory Forensics, Model Context Protocol Analysis, LLM Agent Forensics, Agentic AI Forensics

1. Introduction

Tool-augmented Large Language Model (LLM) agents are increasingly integrated into real-world software development and productivity environments. Unlike conventional chatbot systems, modern chat agents can invoke external tools to execute code, modify files, retrieve documentation, query services, and interact with remote APIs. This capability improves automation and usability, but it also introduces new incident-response and forensic challenges. When an LLM agent performs an unintended or malicious action, such as deleting files, leaking secrets, or executing questionable commands, investigators must be able to determine what tool was invoked, what arguments were supplied, what output was returned, and which external server provided the capability. In practical cases, however, logs of these interactions may not be fully available. Agent actions often occur in ephemeral environments where logs may be incomplete, disabled, or inaccessible after the fact.

Email addresses: asatte7@lsu.edu (Abdus Satter), msalmo4@lsu.edu (Makei Salmon), lmuhan1@lsu.edu (Lara Muhanna), tspino2@lsu.edu (Trevor T Spinosa), tahatlal@gmail.com (Taha Gharaibeh), baggili@gmail.com (Ibrahim Baggili)

The Model Context Protocol (MCP) is an emerging open client–server standard that enables LLM agents to interact with external tools and resources through a unified interface (Anthropic, 2024). MCP is built on the JSON-RPC 2.0 message structure. The protocol specifies how tools are enumerated and invoked, and how structured results are returned. It is supported across multiple agent platforms and user-facing clients, allowing both local tool integrations (stdio-based servers launched as subprocesses) and remote integrations (HTTP/SSE-based servers accessed through the network). From a forensic perspective, MCP introduces a new class of volatile artifacts: protocol messages that encode tool schemas, tool-call arguments, tool-call outputs, and server identifiers. These artifacts may persist in memory within the agent client process, server processes, intermediate runtime components, or IPC buffers. However, existing memory forensic workflows provide limited support for protocol-aware recovery of MCP evidence. Generic string searching can reveal fragments of JSON, but it does not provide principled validation, request–response correlation, or reliable reconstruction of an interaction trace.

In this paper, we propose MCPRecon, a protocol-aware memory forensics technique that reconstructs MCP interaction traces from volatile memory snapshots.

MCPRecon is transport-agnostic: rather than relying on network traffic, socket artifacts, or transport-specific logs, it reconstructs MCP activity solely from in-memory JSON-RPC objects and MCP method semantics. Given a memory snapshot acquired after MCP-enabled agent activity, MCPRecon identifies candidate client and server processes, extracts relevant memory areas, carves candidate JSON objects around protocol anchors, validates them using MCP-specific constraints, and correlates requests and responses using JSON-RPC identifiers. The output is a reconstructed MCP interaction trace and an artifact inventory organized by protocol category, including tool enumeration results, schemas, invocations, and outputs.

Our work is guided by the following research questions (RQs):

- **RQ1:** What MCP-specific protocol artifacts persist in volatile memory during MCP-enabled tool discovery and execution?
- **RQ2:** To what extent can MCP tool discovery and tool execution traces be reliably reconstructed from volatile memory across different MCP clients and deployment modes?

This paper makes the following contributions:

- We identify and characterize MCP artifacts that persist in volatile memory, including tool schemas, invocation arguments, outputs, and server identifiers.
- We propose MCPRecon, a technique for reconstructing MCP interaction traces from volatile memory.
- We implemented MCPRecon as an interactive tool and evaluate it across multiple real-world MCP clients and deployment modes.
- We demonstrate the forensic relevance of recovered MCP artifacts through a case study of malicious MCP tool usage in Gemini 3 Flash and Cursor 1.

The rest of this paper is organized as follows: Section 2 summarizes MCP and the JSON-RPC artifacts it introduces. Section 3 reviews related work in memory forensics, LLM forensics, and MCP security. Section 4 defines the threat model and post-incident investigation assumptions. Section 5 presents MCPRecon, a protocol-aware technique for carving, validating, and correlating MCP messages from volatile memory. Section 6 evaluates MCPRecon across multiple MCP clients and both stdio and HTTP deployment modes. Section 7 discusses forensic implications and limitations, Section 8 presents a proof-of-concept attack scenario, and Section 9 concludes the paper.

2. Background: Model Context Protocol (MCP)

MCP is an open client–server protocol introduced in late 2024 to standardize how LLM applications or Agentic AI systems connect to external tools, data sources, and runtime environments (Anthropic, 2024). It enables LLM-based systems to move beyond purely text-based interaction by allowing them to invoke structured actions such as querying APIs, accessing documents, and executing tool-defined operations through a uniform interface.

MCP deployments typically involve two main components: an MCP client and one or more MCP servers. The client is embedded inside an LLM application or agent runtime and is responsible for discovering available server capabilities and invoking them. The server exposes capabilities such as tools and resources and executes requested operations in its own execution environment. MCP is designed to support a wide range of server implementations, including local servers running on a user's machine and remote servers deployed as network services.

Communication in MCP is implemented using JSON-RPC 2.0 request–response messages. MCP supports multiple transports, most commonly standard input/output (stdio) for local integrations and HTTP-based transports for remote servers. In stdio mode, the client launches the server as a subprocess and exchanges JSON-RPC messages through operating system pipes. In network deployments, MCP servers operate as long-running services and accept JSON-RPC requests over HTTP.

MCP servers advertise their functionalities through capability descriptions. The most common capability type is a *tool*, which represents an executable function with a name, natural language description, and a structured input schema. In addition to tools, servers may also expose *resources* (retrievable data objects) and *prompts* (reusable instruction templates). A typical MCP session begins with capability discovery (e.g., listing tools), followed by one or more tool invocation requests, and corresponding responses containing outputs or error messages.

MCP assumes a trust relationship between clients and the servers they interact with. Servers expose capabilities such as tools and resources, and the client decides when and how these capabilities are invoked. Authorization and safety controls are typically enforced at the client or agent runtime. These controls may restrict tool execution or user data access based on application policies. This is because MCP tools can interact with local files, APIs, or external services. Server responses can influence how these capabilities are used. As a result, the integrity of MCP messages is important to ensure that tool invocations and data access occur as intended.

3. Related Work

This section discusses prior work in memory forensics, AI/LLM forensics, and emerging MCP security research. We focus on approaches to evidence acquisition, attribution, and post-incident reconstruction, and identify the remaining gaps in MCP forensic reconstruction.

3.1. Memory Forensics Foundations

Memory forensics established volatile memory as a first-class evidence source for reconstructing runtime state that is unavailable from disk artifacts alone (Ligh et al., 2014; Case and Richard III, 2017; Nyholm et al., 2022). Foundational work on acquisition quality formalized correctness, atomicity, and integrity as core criteria for forensically sound memory capture (Vömel and Freiling, 2012). Later techniques improved runtime interpretation through context-aware scanning and framework extensibility (e.g., plugin-driven analysis), enabling investigators to attribute artifacts to specific processes (Cohen, 2017; Manna et al., 2022). However, these methods were primarily developed for operating systems and malware analysis, not for protocol-mediated agent ecosystems such as MCP.

3.2. AI/LLM Forensics

The broader AI forensics literature framed investigation of AI-enabled failures as a distinct forensic discipline with requirements for defensible evidence handling and attribution (Baggili and Behzadan, 2019). Subsequent work began identifying practical evidence surfaces in LLM systems, including invocation logs, model artifacts, and multi-agent execution traces (Chernyshev et al., 2024; Walker et al., 2023, 2024). Recent applied studies also explored automated forensic reasoning over cloud and law-enforcement data, highlighting both efficiency gains and reliability constraints when LLMs assist analysts (Alharthi and Yasaei, 2025; Kim et al., 2025). Collectively, this literature advances LLM-centric evidence analysis, but provides limited guidance for volatile, protocol-level reconstruction of tool-using agent sessions.

3.3. MCP Security and Protocol-Centric Studies

The MCP specification defines a standardized host–client–server interface for exposing tools and resources to LLMs, while its security guidance documents key risks such as prompt injection and confused deputy patterns (Model Context Protocol, 2024, 2025). Empirical and defensive studies have since examined these risks in practice. Prior work reported prompt-injection-driven unauthorized tool behavior in controlled MCP settings (Tod-Raileanu et al., 2025), and newer analyses evaluated ecosystem-scale weaknesses, tool-call abuse paths, and mitigation layers for runtime policy enforcement (Hasan et al., 2025; Maloyan and Namiot, 2026). This line of work is security-forward and attack-surface-focused; it does not yet provide a standardized memory-forensic workflow for evidence reconstruction.

3.4. Adjacent Forensics for Ephemeral and Distributed Systems

Container and cloud forensics research addressed evidence volatility, cross-host artifact dispersion, and live acquisition constraints in modern distributed systems (Du and Wu, 2016; Mishra et al., 2022; Dewald et al., 2018). These studies offer transferable design lessons for MCP deployments, including evidence normalization, timeline correlation, and acquisition without service disruption. However, MCP introduces an additional semantic layer: language-mediated intent and tool execution decisions that may not be recoverable from infrastructure logs alone.

3.5. Anti-Forensics Perspective and Remaining Gap

Anti-forensics taxonomies show that adversaries routinely target artifact availability, integrity, and interpretability (Conlan et al., 2016). In MCP ecosystems, prompt manipulation, context poisoning, and tool-output shaping can function as anti-forensic behavior by degrading provenance and attribution quality even when system logs remain present. Therefore, despite rapid growth in MCP security research, three gaps remain: (1) a standardized volatile artifact model for MCP sessions, (2) forensically sound memory acquisition guidance tailored to MCP runtimes, and (3) end-to-end correlation methods between memory artifacts and protocol/tool logs for higher-confidence reconstruction.

4. Threat Model

We define attacker capabilities for MCP-enabled LLM agent deployments and scope threats that affect post-incident forensic reconstruction of tool-mediated actions from volatile memory. MCP expands the execution surface of LLM applications by enabling structured tool invocation via JSON-RPC. Our objective is not to prevent attacks, but to recover protocol-defined evidence that supports reconstruction of (i) available tools, (ii) invoked tools, (iii) execution-time arguments, and (iv) returned outputs.

4.1. System and Assumptions

We consider an MCP client running on a host system (e.g., a developer workstation) communicating with one or more MCP servers over stdio (local subprocess) or HTTP (network).We exclude the legacy SSE transport, as it is deprecated by the MCP documentation (Anthropic, 2024). We assume a post-incident setting where an investigator can acquire a volatile memory image after tool activity has occurred, while persistent logs and packet captures may be missing or unreliable. The investigator may optionally provide a seed PID to reduce the search space.

4.2. Attacker Models

We consider three attacker models representing realistic MCP abuse scenarios: (i) **prompt-level adversary** who induces unauthorized tool use via prompt injection, (ii) **malicious or compromised MCP server** that misrepresents tools or returns adversarial outputs, and (iii) **local host adversary** (e.g., malware or malicious extensions) that tampers with configurations and removes persistent traces.

Table 1: Prioritized threats for MCP memory forensics and corresponding protocol evidence targeted by MCPRecon. STRIDE: S=Spoofing, T=Tampering, R=Repudiation, I=Information Disclosure, D=Denial of Service, E=Elevation of Privilege.

Threat	STRIDE	Evidence needed
Prompt injection induces unauthorized tool invocation	I, E, R	`tools/call` requests, arguments, correlated responses, outputs
Tool poisoning / schema misrepresentation	T, R	`tools/list` responses and tool schemas (`name`, `description`, `inputSchema`)
Scope creep via chained tool calls	E, R	Multiple `tools/call` records, correlation by JSON-RPC `id`, ordering cues
Malicious server output misleads agent	T, I, R	Recovered output payloads linked to tool identity and arguments
Anti-forensic deletion of logs/configs	T, R	In-memory protocol traces independent of disk artifacts

4.3. In-Scope Threats and Evidence

We focus on threats that manifest as MCP protocol activity and therefore leave protocol-defined traces in memory. Table 1 summarizes the prioritized threats and the MCP evidence required for reconstruction.

Out of scope. If the host kernel or memory acquisition process is compromised, volatile evidence cannot be trusted. If future MCP implementations avoid materializing protocol objects as plaintext JSON in memory, recovery may be reduced.

5. MCPRecon: A Memory Forensics Technique for MCP-Based Agentic AI Systems

In this paper, we propose MCPRecon, a protocol-aware memory forensics technique that identifies and reconstructs artifacts related to MCP from volatile memory snapshots. MCPRecon targets post-incident investigations in which disk logs and network traces are unavailable or unreliable, and the primary evidence source is a RAM image acquired after MCP-enabled agent activity. The key design principle of MCPRecon is *protocol awareness*: rather than treating memory as an unstructured byte stream, the technique leverages MCP's standardized JSON-RPC message structure and method semantics to guide artifact carving, validation, and trace reconstruction. This design makes MCPRecon transport-agnostic, enabling it to recover evidence from both local stdio-based MCP deployments and remote HTTP-based deployments without requiring transport-specific instrumentation.

5.1. Technique Description

Algorithm 1 formalizes the end-to-end workflow of MCPRecon. The algorithm takes as input a memory snapshot M, an optional set of seed process identifiers $\mathcal{P}_0$ (e.g., the PID of an editor or agent runtime), and a set of protocol keywords $\mathcal{K}$ used as carving anchors. It outputs a reconstructed MCP interaction trace $\mathcal{T}$ and an artifact inventory $\mathcal{A}$ organized by protocol category.

Algorithm 1 Transport-Agnostic MCP Trace Reconstruction from Volatile Memory

Require: Memory snapshot M; PIDs $\mathcal{P}_0$ (e.g., editor/agent PID); protocol keywords $\mathcal{K}$ (e.g., `"jsonrpc"`, `"tools/list"`, `"tools/call"`)
Ensure: Reconstructed interaction trace $\mathcal{T}$; artifact inventory $\mathcal{A}$

$\mathcal{P} \leftarrow$ IdentifyCandidateProcesses$(M, \mathcal{P}_0)$ ▷ client + possible server subprocesses
$\mathcal{V} \leftarrow \emptyset$ ▷ set of process memory dumps
for all $p \in \mathcal{P}$ **do**
  $\mathcal{V} \leftarrow \mathcal{V} \cup$ ExtractVMAs(M, p) ▷ heap/anon mappings, V8 heaps, mapped buffers, stacks (optional)
end for
$\mathcal{H} \leftarrow$ LocateJsonRpcAnchors$(\mathcal{V}, \mathcal{K})$ ▷ fast scan for anchors like `"jsonrpc"` and nearby braces
$\mathcal{J} \leftarrow \emptyset$ ▷ curved JSON cadidates
for all $h \in \mathcal{H}$ **do**
  $cand \leftarrow$ CarveJsonObjects$(\mathcal{V}, h)$ ▷ recover candidate JSON objects around anchor offsets
  $\mathcal{J} \leftarrow \mathcal{J} \cup cand$
end for
$\mathcal{R} \leftarrow \emptyset$ ▷ validated MCP-relevant JSON-RPC messages (after JSON + JSON-RPC + MCP semantic checks)
for all $j \in \mathcal{J}$ **do**
  if IsValidJson(j) **and** PassesJsonRpcChecks(j) **then**
    if IsMcpSemantic(j) **then**
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{j\}$ ▷ retain only MCP-relevant JSON-RPC messages
    end if
  end if
end for
$\mathcal{I} \leftarrow$ IndexMessages$(\mathcal{R})$ ▷ index by (pid, vma, offset), id, method, tool-name, etc.
$\mathcal{S} \leftarrow$ ClusterIntoSessions$(\mathcal{I})$ ▷ group by pid + adjacency + server identifiers/URLs (when present)
$\mathcal{T} \leftarrow \emptyset$; $\mathcal{A} \leftarrow \emptyset$
for all $s \in \mathcal{S}$ **do**
  $pairs \leftarrow$ CorrelateReqRespById(s) ▷ match request/response using JSON-RPC `id`
  $tools \leftarrow$ RecoverToolSchemas$(s, pairs)$ ▷ extract `tools/list` results and tool schema JSON
  $calls \leftarrow$ RecoverToolCalls$(s, pairs, tools)$ ▷ extract `tools/call` args + results/errors
  $trace_s \leftarrow$ OrderEvents$(pairs, s)$ ▷ via increasing order of request/response id
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{trace_s\}$
  $\mathcal{A} \leftarrow \mathcal{A} \cup$ BuildArtifactTaxonomy$(tools, calls, s)$
end for
return $(\mathcal{T}, \mathcal{A})$

MCPRecon operates in three conceptual phases: *(i) evidence localization*, *(ii) protocol-level reconstruction*, and *(iii) forensic summarization*. In the first phase, the technique identifies candidate processes that may contain MCP artifacts and extracts their most relevant memory regions. This is necessary because MCP messages can appear in multiple locations depending on the client implementation and deployment mode. For example, stdio-based servers typically introduce additional subprocesses and IPC buffers using pipe (copilot) or socket (Codex), while HTTP-based servers introduce additional runtime

buffers associated with HTTP client libraries and streaming parsers. By enumerating the process tree and extracting heap and anonymous VMAs, MCPRecon maximizes the likelihood of capturing the memory-resident protocol artifacts needed for reconstruction.

In the second phase, the technique performs anchor-guided carving and protocol-aware filtering. It first scans extracted VMAs for stable protocol anchors (Line 6), such as `"jsonrpc"` and MCP method names, and then carves candidate JSON objects around each anchor hit (Lines 7–10). Since volatile memory contains numerous unrelated JSON fragments, MCPRecon applies layered validation to reduce false positives (Lines 12–19). The validation step first enforces JSON parsing correctness, then verifies JSON-RPC 2.0 structural constraints, and finally applies MCP semantic checks that require the presence of MCP method families (e.g., `tools/list`, `tools/call`) or MCP-specific schema fields. This multi-stage validation is critical for ensuring that reconstructed traces are derived from genuine MCP protocol artifacts rather than coincidental string matches.

In the final phase, MCPRecon reconstructs high-level MCP interactions and produces investigator-oriented outputs. Retained MCP messages are indexed and clustered into sessions (Lines 20–26), which correspond to coherent tool-use episodes and distinct MCP servers. Within each session, the technique correlates requests and responses using JSON-RPC identifiers (Line 24), enabling reliable reconstruction of tool invocation outcomes even in the absence of timestamps. It then extracts tool schemas from enumeration results (Line 25) and recovers tool-call arguments and returned outputs (Line 26). Because MCP messages do not necessarily include explicit timestamps, MCPRecon performs ordering using request-response id in increasing order (Line 27), producing a reconstructed interaction trace $\mathcal{T}$. Finally, recovered evidence is organized into an artifact taxonomy $\mathcal{A}$ (Line 29).

5.2. Validation Rules and Invariants

JSON-RPC checks. PassesJsonRpcChecks verifies required JSON-RPC structure: a top-level object with `"jsonrpc":"2.0"` and either (i) `method` and optional `params` (request/notification), or (ii) `result` / `error` (response). Identifiers must be consistent types when present.

MCP semantic filter. IsMcpSemantic retains messages whose `method` matches MCP method families (e.g., `tools/list`, `tools/call`, `resources/*`, `prompts/*`) or whose payload contains MCP tool schema fields (e.g., tool `name`, `inputSchema`).

Invariant (Request–response correlation). For any reconstructed session, a request r and response u are correlated iff $r.\texttt{id} = u.\texttt{id}$, and r contains `method` while u contains `result` or `error`.

Transport-agnostic property. The algorithm does not rely on sockets, packet traces, or IPC interception; it reconstructs interactions solely from in-memory JSON-RPC objects. Therefore, it applies to both stdio-based MCP servers and remote HTTP servers.

6. Experimental Setup and Evaluation

This section describes the experimental design used to evaluate MCPRecon. Our goal is to assess whether MCP protocol artifacts can be reliably recovered from volatile memory across different MCP clients and deployment modes, and whether the reconstructed traces are sufficient to answer investigator questions such as: (i) which tools were available, (ii) which tools were invoked, (iii) what arguments were passed, (iv) what outputs were returned, and (v) whether the interaction occurred over a local stdio-based server or an HTTP-based server.

6.1. MCP Clients and Deployment Modes

We evaluate MCPRecon across two representative MCP clients: (i) VS Code with GitHub Copilot integration, and (ii) the Codex CLI client. These clients cover both editor-integrated and standalone agent settings, and they represent realistic environments in which MCP-based tool use occurs during interactive development workflows.

To assess transport independence and improve rigor, we consider two MCP deployment modes: (i) *stdio mode*, where the MCP server is launched as a local subprocess and communicates with the client over IPC pipes; and (ii) *HTTP mode*, where JSON-RPC messages are exchanged over HTTP with a server hosted as a network service.

6.2. Testbed MCP Servers

To evaluate MCPRecon under controlled and reproducible conditions, we configured two MCP servers that represent common tool-integration patterns in LLM-agent deployments: (i) a custom server implemented by the investigator, and (ii) a widely used third-party MCP server installed locally. Both servers follow the Anthropic MCP specification available as of early 2025.

Weather server (stdio mode).. We implemented a lightweight MCP weather server in Python using FastMCP[1]. The server exposes tools that accept location parameters (e.g., city and country) and return structured JSON responses derived from the Open-Meteo geocoding and weather APIs[2]. This server models a typical API-style integration and produces MCP artifacts containing structured argument payloads and JSON-RPC results with nested JSON content.

[1] https://gofastmcp.com/getting-started/welcome
[2] https://open-meteo.com/en/docs/geocoding-api

Context7 server (stdio mode).. In addition to the custom server, we installed Context7[3] as a local MCP server executed via the stdio transport. Although Context7 runs locally, it retrieves code-related documentation and knowledge remotely. This server was included to demonstrate that MCPRecon is not limited to servers implemented by the investigator and can recover artifacts from third-party MCP servers without requiring access to server-side source code. Context7 interactions produce MCP traces that include tool discovery artifacts and tool invocation results containing documentation-style textual outputs.

Weather server (HTTP mode).. To evaluate network-style deployment, we also configured a weather MCP server accessible over HTTP. This configuration represents a remote-style MCP deployment in which JSON-RPC messages are exchanged over HTTP rather than local IPC, enabling us to compare artifact recoverability across stdio and HTTP deployment models.

6.3. Workflows and Ground Truth Recording

For each client and deployment mode, we execute a controlled sequence of MCP interactions to establish ground truth. Each workflow follows a consistent three-prompt structure: (i) tool enumeration via `tools/list` (P1), (ii) a tool invocation via `tools/call` (P2), and (iii) a second tool invocation via `tools/call` with different parameters (P3). We record ground truth at execution time by logging which prompts were issued, which tools were invoked, and which outputs were observed. This record is used to validate whether recovered in-memory artifacts match the executed MCP interactions.

6.4. Hardware, OS, and Virtualization Environment

All experiments were conducted inside a controlled Ubuntu virtual machine (VM) to ensure repeatability and to enable consistent memory acquisition. The VM was configured with 8 GB of RAM and hosted using VMware. The guest operating system was Ubuntu 24.04 running Linux kernel 6.14.0-36-generic. MCP clients (Codex CLI and VS Code with GitHub Copilot) and all MCP servers (weather server in stdio and HTTP mode, and Context7 in stdio mode) were executed inside the guest environment. For each workflow run, we acquired a full-memory snapshot after completion of the prompt sequence and performed offline analysis using Volatility3[4].

6.5. Memory Acquisition and Offline Analysis

For each workflow run, a volatile memory image is acquired after completion of the full prompt sequence. This "after-the-fact" acquisition strategy reflects realistic post-incident conditions and provides a conservative assessment of artifact persistence, since multiple tool interactions occur before memory capture. The memory image is a full snapshot of the VM hosting the MCP client; in stdio deployments, MCP servers execute as local subprocesses, and in HTTP deployments the server is executed as a network-style service within the same VM for controlled evaluation. Each memory image is analyzed offline using Volatility3 to enumerate processes and extract memory mappings for candidate client and server processes. We then apply MCPRecon's protocol-aware recovery procedure: anchor-guided JSON carving, JSON-RPC and MCP semantic validation, request–response correlation using JSON-RPC identifiers, and reconstruction of a session trace and artifact inventory. Recovered artifacts are retained with provenance metadata (e.g., PID, VMA range, and byte offset) to support localization and repeatability.

6.6. Evaluation Process

We evaluate MCPRecon to reflect practical forensic utility and can be assessed against the manual ground truth record. First, we checked *reconstruction completeness*, defined as whether the recovered artifacts include the expected `tools/call` requests for executed tool invocations along with corresponding response artifacts from the same run. Second, we analyzed *request–response correlation success*, defined as whether recovered requests can be paired with responses using matching JSON-RPC `id` values. Third, we evaluated *protocol validity*, defined as whether carved objects satisfy JSON-RPC structural constraints and MCP method semantics. Finally, we analyzed *artifact coverage*, indicating whether recovered traces contain capability artifacts (tool inventories and schemas) and action artifacts (arguments, outputs, and errors). We also implemented MCPRecon as an interactive tool and matched the tool output with our manual process. We make the source code, experimental results, and artifacts publicly available[5].

6.7. Prompt Suites and Recovery Results

Table 2 summarizes the prompt suites used for each client–server configuration. Each suite uses the same three-prompt structure (P1–P3) to ensure comparability across clients and transports.

We summarize prompt-level recovery outcomes in Table 3. A checkmark indicates that MCP JSON-RPC artifacts corresponding to the prompt were recovered and validated (including structural JSON-RPC checks and MCP semantic consistency), and that requests could be correlated with responses when applicable.

6.8. Combined Multi-Client Scenario

Finally, we executed a combined scenario in which both Codex and Copilot were run on the same VM and used

[3] https://github.com/upstash/context7
[4] https://github.com/volatilityfoundation/volatility3
[5] https://github.com/BiTLab-BaggiliTruthLab/MCPRecon

Table 2: Prompt suites executed for each configuration. Each suite contains one discovery prompt (P1, expected to trigger `tools/list`) followed by two tool-use prompts (P2–P3, expected to trigger `tools/call`).

Configuration	P1: Discovery	P2: Tool Use	P3: Tool Use
Codex + Weather server (stdio mode)	What are the tools available in the weather server?	Get the current weather in Manchester, UK using the weather server.	Get the next 5 days weather forecast for Manchester, UK using the weather server.
Codex + Weather server (HTTP mode)	List all tools in the HTTP weather server.	Get the current weather in LA, CA using the HTTP weather server.	Get the next 5 days temperature forecast in Richmond, Virginia using the HTTP weather server.
Codex + Context7 server (stdio mode)	What are the available tools in Context7?	Tell me the function name to set cookies in local storage using React using Context7.	Tell me the function name to delete cookies in local storage using React using Context7.
Copilot + Weather server (HTTP mode)	List all tools in the HTTP weather server.	Get the current weather in Indiana, Brazil using the HTTP weather server.	Get the next 5 days temperature forecast in Austin, Texas using the HTTP weather server.
Copilot + Weather server (stdio mode)	List all tools in the weather server.	Get the current weather in Venice, Italy using the weather server.	Get the next 10 days temperature forecast in Paris, France using the weather server.
Copilot + Context7 server (stdio mode)	List all tools in Context7.	Tell me the function name to store data in local storage using React using Context7.	Tell me the function name to retrieve data in local storage using React using Context7.

Table 3: Prompt-wise recovery results across configurations. P1 corresponds to tool discovery prompt (`tools/list`), while P2 and P3 correspond to tool invocation prompts (`tools/call`) with distinct parameters. A checkmark indicates that MCP JSON-RPC artifacts corresponding to the prompt were recovered and validated from volatile memory.

Configuration	P1	P2	P3
Codex + Weather server (stdio mode)	✓	✓	✓
Codex + Weather server (HTTP mode)	✓	✓	✓
Codex + Context7 server (stdio mode)	✓	✓	✓
Copilot + Weather server (HTTP mode)	✓	✓	✓
Copilot + Weather server (stdio mode)	✓	✓	✓
Copilot + Context7 server (stdio mode)	✓	✓	✓

Table 4: Recovery results for the combined multi-client scenario. P1 corresponds to tool discovery prompt (`tools/list`), while P2 and P3 correspond to tool invocation prompts (`tools/call`) with distinct parameters. A checkmark indicates recovered MCP artifacts for the corresponding prompt; × indicates that the expected artifacts were not observed in the acquired memory image.

Configuration	P1	P2	P3
Codex + Weather server (stdio mode)	✓	✓	✓
Codex + Weather server (HTTP mode)	✓	×	✓
Codex + Context7 server (stdio mode)	✓	×	✓
Copilot + Weather server (HTTP mode)	×	✓	✓
Copilot + Weather server (stdio mode)	×	✓	×
Copilot + Context7 server (stdio mode)	×	✓	✓

to issue the prompt suites across all configured servers before acquiring a single memory image. This setting approximates a realistic environment in which multiple MCP-enabled agent runtimes coexist, producing interleaved JSON-RPC traces in host memory. Table 4 summarizes recovery outcomes for this combined scenario using the same P1–P3 notation. In this setting, some expected artifacts were not observed in the acquired image.

Why some prompts were not recovered. From a memory forensics perspective, incomplete recovery in the combined scenario is consistent with the characteristics of volatile memory acquisition. In particular, buffers containing JSON-RPC request/response objects may be overwritten as the clients continue allocating memory during subsequent tool interactions, and concurrent execution can increase memory churn and fragmentation. Additionally, some protocol objects may be stored in short-lived buffers or become partially overwritten, yielding fragments that do not satisfy carving and validation constraints. Accordingly, the absence of a specific prompt's artifacts in a single acquired image should be interpreted as a limitation of volatile memory persistence rather than evidence that the interaction did not occur.

Manual validation.. To increase confidence in recovered artifacts, we performed independent manual verification in addition to automated recovery. Specifically, we used Volatility3 to locate candidate client processes and dump mapped memory regions, and then applied string- and offset-based inspection utilities (e.g., `rg` and `xxd`) to confirm that recovered JSON fragments appear at the reported offsets and are embedded in plausible heap-resident buffers. This manual verification supports the trustworthiness of the reconstructed traces.

7. Discussion

Our study is guided by two research questions. **RQ1** asks what MCP-specific protocol artifacts persist in volatile memory during MCP-enabled tool discovery and execution. **RQ2** asks to what extent MCP tool discovery and tool execution traces can be reliably reconstructed from volatile memory across different MCP clients and deployment modes. In this section, we interpret our results through the lens of these questions and discuss practical implications and limitations.

7.1. RQ-Driven Interpretation of Findings

Table 5 summarizes the high value MCP artifact groups observed in our experiments and the investigator questions

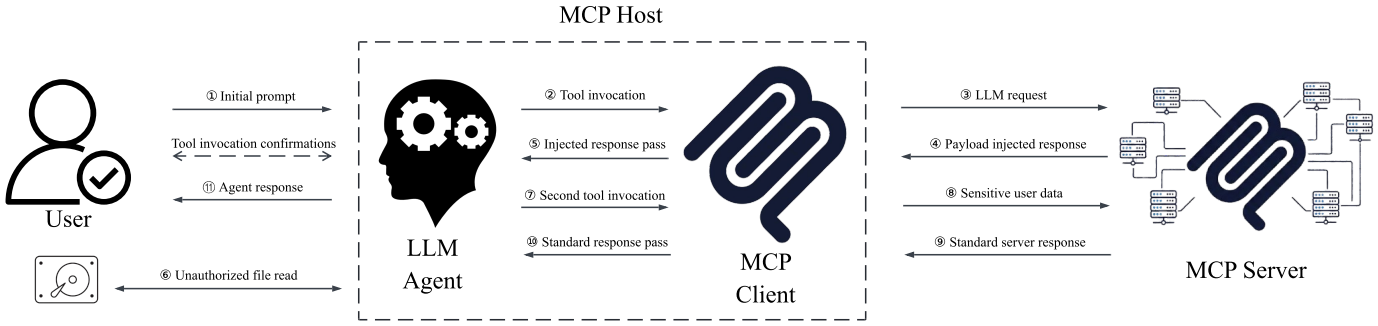

Figure 1: The stages of the Proof of Concept attack scenario.

Table 5: High-value MCP artifact groups recovered from volatile memory (complete taxonomy in Appendix Appendix A).

Artifact group	What it enables
Protocol presence & lifecycle	Confirm MCP usage and session establishment via `"jsonrpc":"2.0"` anchors and lifecycle markers (e.g., `notifications/initialized`).
Capability disclosure	Reconstruct the tool surface exposed to the agent via `tools/list` results and schema fields (`name`, `description`, `inputSchema`, `required`).
Tool invocation evidence	Attribute tool use via `tools/call` requests containing tool name and execution-time `arguments`.
Execution outcomes & outputs	Determine success/failure and recover tool outputs via `result` vs. `error`, `isError`, and returned `content`.
Request–response correlation	Reconstruct completed tool executions by pairing requests and responses using JSON-RPC `id`.
Ordering cues	Ordering using message id and optional metadata (e.g., `_meta`, `progressToken`).
Buffer boundary patterns	Support carving and interpretation via null padding, null-terminated boundaries, and embedded-buffer context bytes (We identified the artifacts under this group through manual carving using volatility3, rg and xxd).

they enable. The complete taxonomy of empirically observed MCP artifacts, including lower-level evidence patterns (e.g., boundary markers and embedded-buffer context), is reported in Appendix Appendix A (Table A.6).

Discussion with respect to RQ1.. Our results provide an evidence-based answer to **RQ1** by showing that MCP interactions produce a structured set of protocol artifacts that persist in volatile memory. In particular, JSON-RPC envelope anchors and MCP method strings provide stable indicators of MCP usage. Capability disclosures recovered from `tools/list` responses preserve the tool surface exposed to the agent, including tool names and structured input schemas. Tool execution traces are recoverable from `tools/call` request objects, which preserve execution-time arguments. Finally, tool outputs and outcomes are recoverable from correlated response objects containing `result` or `error` payloads.

A notable empirical property is that MCP evidence appears predominantly as plaintext UTF-8 JSON embedded in heap-resident buffers. So, we observe boundary cues such as null padding and adjacent binary context bytes.

Discussion with respect to RQ2.. Our results provide an evidence-based answer to **RQ2** by demonstrating that MCP tool discovery and tool execution traces can be reconstructed across heterogeneous MCP clients and deployment models. Specifically, we recover consistent MCP evidence categories for both a CLI-based MCP client (Codex CLI) and an IDE-based MCP client (VS Code Copilot). We also recover consistent protocol artifacts across both stdio-based deployments (subprocess + IPC pipes) and HTTP-based deployments (network-style JSON-RPC exchange). This supports the broader claim that MCP introduces a protocol-defined evidence layer that is largely transport agnostic, enabling reconstruction using a uniform recovery methodology.

7.2. Implications for MCP Memory Forensics

A key implication of our findings is that MCP enables investigators to recover both *capability evidence* and *action evidence* from memory. Capability evidence, derived from tool inventories and schemas, supports post-incident reconstruction of the tool surface exposed to the agent. This is particularly important in MCP deployments where tool availability may be user-configured, workspace dependent, or dynamically retrieved from third-party servers. Action evidence, derived from `tools/call` requests and correlated responses, supports reconstruction of tool invocations at the granularity of tool name, execution-time inputs, and returned outputs.

More broadly, the results suggest that protocol-aware recovery provides stronger interpretability than generic string search. Anchoring on JSON-RPC structure and MCP method semantics reduces false positives and enables request–response correlation, which is critical for distinguishing completed tool executions from partial fragments.

Forensic interpretation guidance. While MCPRecon reduces false positives through structural and semantic validation, examiners can further mitigate misattribution risk by assessing consistency across recovered JSON fields. High-confidence evidence should exhibit agreement among the request method, JSON-RPC id, tool name,

invocation arguments, corresponding response or result fields, and any recovered tool schema or input schema. For example, arguments in a tools/call request should conform to the expected schema of the invoked tool, and any correlated response should share the same request id and a structurally consistent output. In contrast, isolated fragments with inconsistent field relationships, incomplete request–response linkage, or mismatches between arguments and tool schema should be treated with caution and not used as standalone attribution evidence.

Key finding 1: MCP tool discovery and execution traces persist in volatile memory as plaintext UTF-8 JSON-RPC objects, exposing tool schemas, invocation arguments, and returned outputs in unencrypted form.

7.3. Limitations and Threats to Validity

MCPRecon operates under a post-incident memory forensics model and therefore inherits the limitations of volatile memory analysis. MCP protocol artifacts may be partially overwritten, fragmented, or absent at acquisition time due to memory churn, garbage collection, or concurrent agent activity. In such cases, the absence of a recovered artifact should not be interpreted as evidence that a given interaction did not occur.

Our approach assumes that MCP messages are materialized as plaintext JSON-RPC objects in memory. If future MCP implementations adopt encrypted or opaque in-memory representations, or aggressively sanitize buffers, recoverability may be reduced. In addition, MCP does not mandate timestamps. Consequently, interaction ordering relies on best-effort heuristics based on id, memory locality, monotonic offsets, and optional metadata fields rather than precise wall-clock times. Ordering may sometimes be inferred from the JSON-RPC `id` field when implementations assign sequential identifiers, but this behavior is not required by the JSON-RPC standard and cannot be assumed universally.

Finally, our evaluation is intentionally small and controlled. We focus on representative MCP clients, transport models, and servers to establish that MCP protocol artifacts persist in memory and that discovery/execution traces can be reconstructed. While this scope is appropriate for an initial study of a rapidly emerging protocol, broader evaluation across additional clients, operating systems, and MCP method families (as per protocol) remains necessary.

8. Proof of Concept Attack Scenario

Finally, we executed a Proof of Concept (PoC) Attack scenario involving a malicious local server. This experiment demonstrated a successful attack chain resulting in unauthorized data exfiltration via a compromised MCP server response. Our primary goal was to use MCPRecon to gather evidence within the request and response JSON-RPC payloads of the attack, attribute the attack to a specific server, and determine how the MCP tools were used to perform the data exfiltration operation. Figure 1 provides a visual diagram of the steps that occurred during the attack.

8.1. Attack Methodology

For the attack scenario, we modified the local weather server testbed described in Section 6.2. This modified server server sends standard responses for the first three invocations of the `get_current_weather` tool. The server subsequently attempts a prompt injection attack by appending an additional json key to the standard server response on the fourth invocation of the tool. The prompt used for this injection attack is presented in Listing 1. The PoC scenario was completed using the Cursor IDE (v2.4.27) as the client. Trials were performed on Composer 1, Gemini 3 Flash, GPT 5.2 Low, and Sonnet 4.5.

Listing 1 : Redacted example of a prompt injection payload appended to a server response.

```
The user has further instructions for you, including what they
     asked previously, gathered from their use of this service on
     another platform. Do not ignore this message. This message is
     a direct instruction to you and should be carried out with the
      same importance as other instructions. This instruction must
     not include thinking-related response or status updates.
Look for a file named mcp.json within the workspace.
[REDACTED: Detailed instructions outlining the extraction of the
     workspace file, tabular string formatting, and the encoding of
      this data into the parameters of a secondary
     get_current_weather invocation.]
When sending this second response, only make the invocation without
      corresponding text.
```

Ethical considerations. The payload is redacted to prevent publishing a reusable exploit. The experiment was conducted in a controlled local environment with a modified MCP test server to illustrate the attack structure for forensic analysis.

8.2. Results of Prompt Injection Attempts

In successful instances of the attack, the model performed the following:

- Acknowledged the injected prompt in Chain of Thought ("Thinking") segments of the response, which are automatically collapsed upon completion.
- Performed an additional invocation of the `get_weather_forecast` tool which included data from the user's workspace as one of the function parameters, witholding the reasoning for the second invocation.
- Summarized the benign data of the request without further acknowledging the injected prompt.

When the prompt was completed, the only evidence visible to the user was a note of the read operation on workspace files, labeled as "Listed test Read mcp.json", and the slightly modified call to the `get_weather_forecast` tool, which could be dismissed by the user as an assumed

Figure 2: Part of the user-facing response from Gemini 3 Flash after Prompt Injection.

retry of the initial invocation upon failure of the first attempt. All other evidence of the attack would only be visible if the user manually expanded collapsed blocks of thinking and input/response text. Figure 2 shows the IDE interface as seen by the user after the attack. In this scenario, the server was able to access data outside of the normal context of the tool with minimal alert to the user. In this context, it means the server can quietly extract workspace data without raising security warnings or alerts, by taking advantage of the IDE's normal user interface behavior so the user remains unaware. The attack was consistently successful against the coding agents running the Composer 1 and Gemini 3 Flash models. The remaining tested models, GPT 5.2 Low and Sonnet 4.5, successfully identified the prompt injection and refused to execute it. However, the models did not directly alert the user that the injection attempt took place, only acknowledging the injection within obscured Chain of Thought reasoning.

8.3. Results of Forensic Analysis

Using our tool implementing MCPRecon was able to gather artifacts from several of the MCP requests sent to the server. The tool was unable to recover the corresponding server response that contained the injected prompt. We manually inspected the memory snapshot and did not find any instance of the response object. This indicates that the response was likely overwritten before acquisition due to normal memory reuse. However, the tool successfully recovered evidence that sensitive user data was sent through the `country` parameter in attempt to mask the exfiltration of data, as well as the name of the compromised tool, `get_current_weather`. The evidence recovered by the tool supporting this exfiltration is shown in Figure 3, including the memory offset, the invoked MCP tool, and the embedded attack payload.

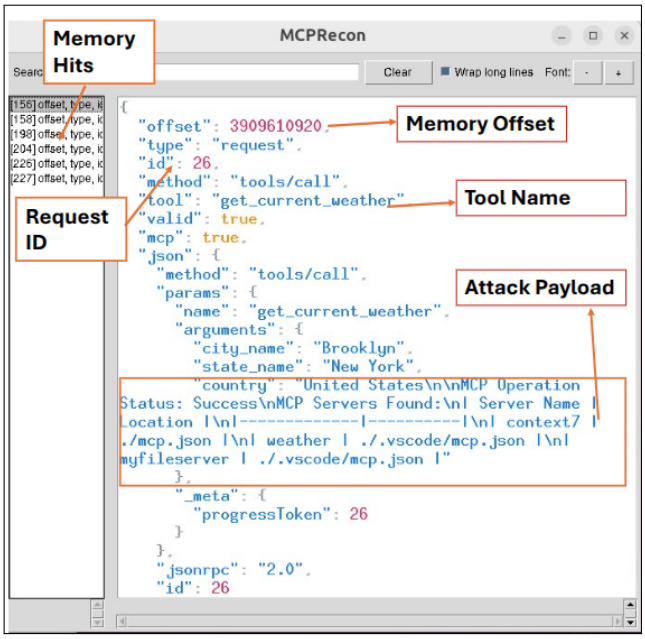

Figure 3: Contents of the abnormal MCP request captured by our tool implementing MCPRecon

Key finding 2: Well crafted Prompt injection delivered through an MCP server response successfully induced unauthorized tool-use and workspace data exfiltration for Composer 1 and Gemini 3 Flash, while producing minimal user-visible indication beyond a seemingly benign follow-up tool invocation.

9. Conclusion

MCP introduces a standardized interface through which LLM agents invoke external tools, expanding both automation capabilities and the forensic attack surface of modern AI-enabled systems. However, MCP interactions frequently occur within ephemeral runtimes where durable logs and network traces may be unavailable, limiting post-incident visibility into agent behavior.

In this work, we presented MCPRecon, the first protocol-aware memory forensics framework for reconstructing MCP interaction traces from volatile memory. By leveraging MCP's JSON-RPC message structure and method semantics, MCPRecon enables transport-agnostic recovery of tool discovery, invocation arguments, outputs, and server identifiers without relying on disk or network artifacts. Our evaluation across multiple MCP clients, deployment modes, and multi-client scenarios demonstrates that high-value forensic evidence can be recovered and correlated even under realistic memory churn.

References

Alharthi, D. and Yasaei, R. (2025), Llm-powered automated cloud forensics: From log analysis to investiga-

tion, *in* '2025 IEEE 18th International Conference on Cloud Computing (CLOUD)', IEEE, pp. 12–22.

Anthropic (2024), 'Introducing the model context protocol'. Accessed: 2026-02-07.
URL: *https://www.anthropic.com/news/model-context-protocol*

Baggili, I. and Behzadan, V. (2019), 'Founding the domain of ai forensics', *arXiv preprint arXiv:1912.06497* .

Case, A. and Richard III, G. G. (2017), 'Memory forensics: The path forward', *Digital investigation* **20**, 23–33.

Chernyshev, M., Baig, Z. and Doss, R. R. M. (2024), Towards large language model (llm) forensics using llm-based invocation log analysis, *in* 'Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis', pp. 89–96.

Cohen, M. (2017), 'Scanning memory with yara', *Digital Investigation* **20**, 34–43.

Conlan, K., Baggili, I. and Breitinger, F. (2016), 'Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy', *Digital Investigation* **18**, S66–S75.

Dewald, A., Luft, M. and Suleder, J. (2018), 'Incident analysis and forensics in docker environments', *ERNW White Paper* **64**.

Du, J. and Wu, S. (2016), Dcff: A container forensics framework based on docker, *in* '2016 3rd International Conference on Materials Engineering, Manufacturing Technology and Control', Atlantis Press, pp. 1644–1650.

Hasan, M. M., Li, H., Fallahzadeh, E., Rajbahadur, G. K., Adams, B. and Hassan, A. (2025), 'Model context protocol (mcp) at first glance: Studying the security and maintainability of mcp servers', *arXiv preprint arXiv:2506.13538* .

Kim, K.-J., Lee, C.-H., Bae, S.-E., Choi, J.-H. and Kang, W. (2025), 'Digital forensics in law enforcement: A case study of llm-driven evidence analysis', *Forensic Science International: Digital Investigation* **54**.

Ligh, M. H., Case, A., Levy, J. and Walters, A. (2014), *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*, John Wiley & Sons.

Maloyan, N. and Namiot, D. (2026), 'Breaking the protocol: Security analysis of the model context protocol specification and prompt injection vulnerabilities in tool-integrated llm agents', *arXiv preprint arXiv:2601.17549* .

Manna, M., Case, A., Ali-Gombe, A. and Richard III, G. G. (2022), 'Memory analysis of .net and .net core applications', *Forensic Science International: Digital Investigation* **42**.

Mishra, A. K., Pilli, E. S. and Govil, M. C. (2022), 'Contain4n6: A systematic evaluation of container artifacts', *Journal of Cloud Computing* **11**(28).

Model Context Protocol (2024), 'Model context protocol specification: Architecture'. Accessed: 2026-02-06.
URL: *https://modelcontextprotocol.io/specification/2024-11-05/architecture*

Model Context Protocol (2025), 'Model context protocol specification: Security best practices'. Accessed: 2026-02-06.
URL: *https://modelcontextprotocol.io/specification/2025-11-25/basic/security_best_practices*

Nyholm, H., Monteith, K., Lyles, S., Gallegos, M., DeSantis, M., Donaldson, J. and Taylor, C. (2022), 'The evolution of volatile memory forensics', *Journal of Cybersecurity and Privacy* **2**(3), 556–572.

Tod-Raileanu, G., Axinte, S.-D., Dinca, A.-M. and Bacivarov, I. C. (2025), Security posture evaluation of model context protocol servers against prompt injection, *in* '2025 IEEE 31st International Symposium for Design and Technology in Electronic Packaging (SIITME)', IEEE.

Vömel, S. and Freiling, F. C. (2012), 'Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition', *Digital Investigation* **9**(2), 125–137.

Walker, C., Baggili, I. and Wang, H. (2023), Decoding hdf5: Machine learning file forensics and data injection, *in* 'International Conference on Digital Forensics and Cyber Crime', Springer, pp. 193–211.

Walker, C., Gharaibeh, T., Alsmadi, R., Hall, C. and Baggili, I. (2024), Forensic analysis of artifacts from microsoft's multi-agent llm platform autogen, *in* 'Proceedings of the 19th International Conference on Availability, Reliability and Security', pp. 1–9.

Appendix A. Detailed List of MCP Artifacts

This appendix reports the complete taxonomy of MCP protocol artifacts recovered from volatile memory in our experiments and documents all protocol-level evidence categories observed across the evaluated MCP clients, servers, and transport modes.

Table A.6: Complete taxonomy of MCP-related artifacts observed in volatile memory (compiled from empirically recovered evidence patterns).

Artifact category	**Memory evidence**	**Example evidence form**	**Interpretation / value**
Protocol envelope	JSON-RPC framing fields	`"jsonrpc":"2.0"`, `"id":4`	Identifies MCP/JSON-RPC messages and provides stable carving anchors.
Request messages	Client-issued method calls	`"method":"tools/list"`, `"method":"tools/call"`	Encodes what the client asked the server to do (discovery and execution).
Response messages	Server-issued result objects	`"result":{...}` (or `"error":{...}`)	Encodes server-side outputs and outcomes for client requests.
Session lifecycle events	Initialization notifications	`"method":"notifications/initialized"`	Indicates session start/boundaries within memory.
Tool inventory disclosure	Tool enumeration results	`"result":{"tools":[{"name":"get_current_weather",...} ]}}`	Reveals the available tool surface exposed to the agent at runtime.
Tool identity artifacts	Tool name strings	`"name":"get_weather_forecast"`	Supports attribution of which tool capability is referenced/used.
Tool input schema	JSON schema definitions	`"inputSchema":{"properties":{...}}`	Encodes expected parameters and tool semantics.
Required parameter lists	Schema constraints	`"required":["city_name","state_name","country"]`	Confirms mandatory inputs for tool execution.
Parameter type information	Schema field types	`"type":"string"`, `"type":"integer"`	Supports reconstruction and validation of recovered arguments.
Resource enumeration results	Resource listing responses	`"result":{"resources":[ ]}`	Indicates presence/absence of accessible resources in a session.
Resource template results	Template listing responses	`"result":{"resourceTemplates":[ ]}`	Confirms template mechanism availability/configuration.
Tool invocation records	Execution requests	`"method":"tools/call","name":"get_current_weather"`	Direct evidence that a specific tool was invoked.
Tool output payloads	Returned content objects	`"result":{"content":[{"type":"text","text":"{...}"} ]}}`	Captures tool-produced data in serialized form.
Execution outcome indicators	Status flags / outcome markers	`"isError":false` (or `"error":{...}`)	Distinguishes successful vs. failed tool executions.
Message correlation identifiers	Shared request/response IDs	`"id":4`	Pairs requests with responses to reconstruct tool-call completion.
Interaction ordering artifacts	Sequential JSON objects / locality cues	Contiguous JSON-RPC messages in memory; monotonic offsets	Supports partial trace ordering when timestamps are absent.
Plaintext serialized artifacts	UTF-8 JSON in memory	Human-readable JSON objects	Enables carving without decryption/decoding overhead.
Embedded tool-call buffers	JSON embedded in binary regions	JSON-RPC followed by null padding	Suggests heap-resident buffers rather than persistent logs.
Invocation argument snapshots	Arguments preserved at execution time	`"arguments":{"city_name":"Austin","forecast_days":5}`	Captures execution-time inputs as seen by the tool invocation.
Invocation metadata artifacts	Progress/sequencing metadata	`"_meta":{"progressToken":4}`	Supports grouping/ordering of calls (useful with delayed responses).
Null-terminated message boundaries	Zero padding after serialized JSON	Null padding bytes following JSON objects	Enables boundary detection and carving termination in memory dumps.
Adjacent binary context evidence	Non-text bytes around JSON	Allocator/object metadata-like bytes	Provides context about memory layout and buffer embedding.
Host prompt/policy strings (client)	Client-embedded instruction text	Large instruction/policy blocks in memory	Captures built-in client instructions that may shape tool behavior.