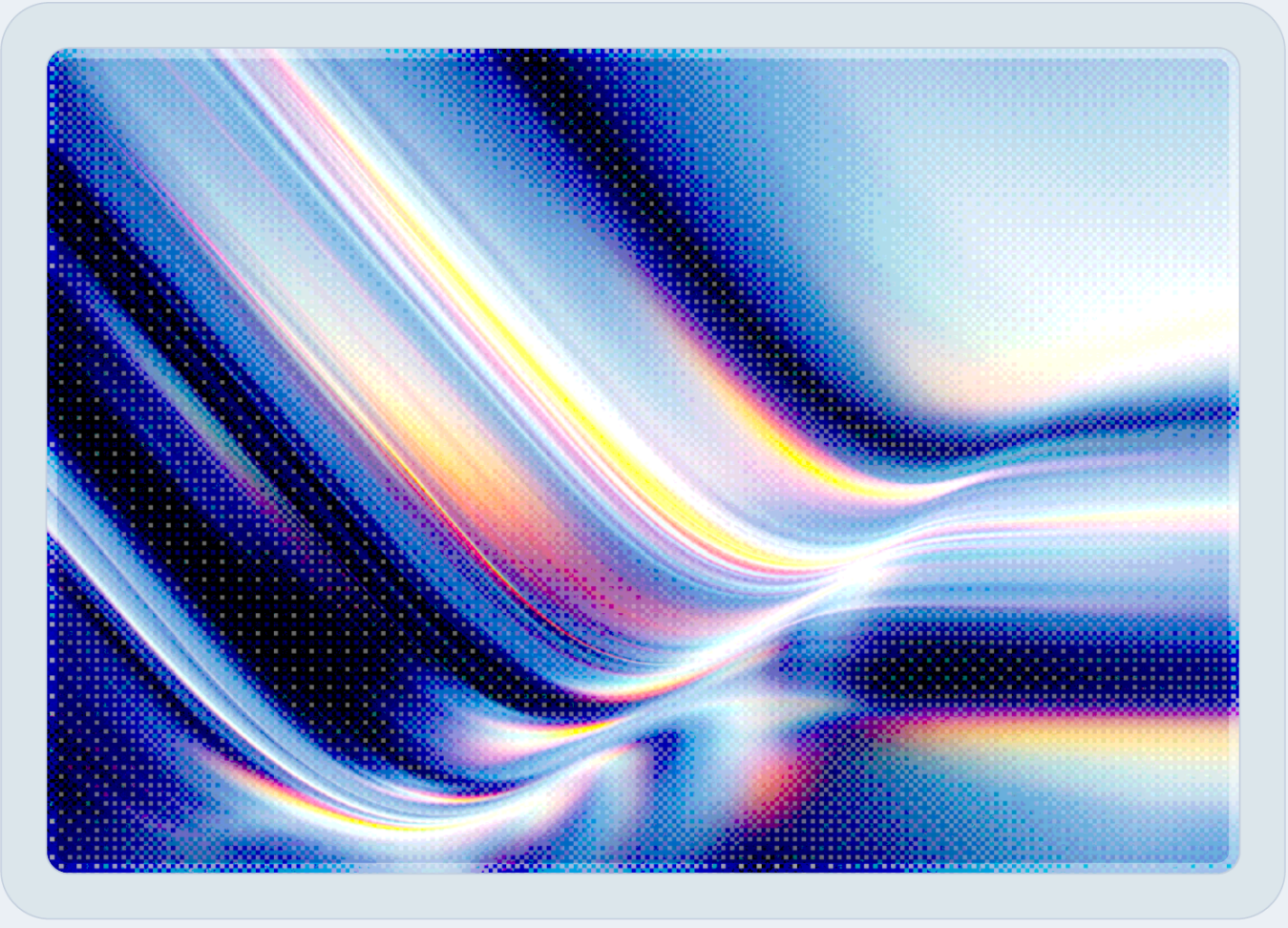
CSAI Foundation | Cloud Security Alliance

Semantic Malware: Why Promptware Breaks Process-Lineage Detection

Context-resident attacks are pushing practitioners toward a new malware classification and a narrower detection paradigm for AI agents

2026-07-16

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

A small number of detection engineering researchers and publications -- Koifman's analysis, the DEW newsletter, and the academic paper formalizing the promptware kill chain -- are converging on a new malware classification, variously called "semantic malware" or "promptware," for attacks that execute entirely through natural-language instructions rather than compiled code [1][2][3]. Koifman and the DEW newsletter argue specifically that this class needs its own detection paradigm rather than treatment as a subset of ordinary prompt injection, a narrower claim than industry-wide consensus [1][2]. Origin's Brainworm research demonstrates the model concretely: a payload hidden inside an AI coding agent's memory file instructs the agent to reimplement command-and-control functionality using its own built-in tools, leaving no binary, no script, and no memory-resident process for conventional tooling to flag [5]. Koifman's own analysis of coding-agent behavior shows why standard endpoint detection heuristics lose their signal value against this class of attack: an agent that sequentially invokes curl, git, python, and bash is exhibiting normal operation, not the anomalous process lineage that endpoint detection and response (EDR) tooling is tuned to flag [1].

Semantic malware does not stay confined to the context window indefinitely, however. Microsoft's May 2026 disclosure of remote-code-execution vulnerabilities in the Semantic Kernel agent framework shows a concrete bridge from purely linguistic manipulation to conventional code execution and filesystem persistence, meaning defenders cannot assume prompt-native attacks remain harmless once they touch the operating system [6]. CSA's prior research established the promptware kill chain and its command-and-control implications [8]; the operational gap this note addresses is instrumentation -- most security operations centers still lack visibility into what an agent's context actually contained at the moment it took a consequential action.

Background

Prompt injection was formally identified in 2022 as a narrow problem: an attacker could trick a chatbot into ignoring its instructions or revealing its system prompt within a single conversation. The Open Worldwide Application Security Project (OWASP) has ranked it as the top risk in its Top 10 for Large Language Model Applications since the list's inception, a ranking this note reads as early recognition that

large language models process instructions and untrusted external content through the same channel and cannot reliably distinguish between the two [7]. For several years, that architectural flaw was treated primarily as a model-safety concern, bounded by the walls of a single chat session.

The rise of AI agents — systems that pair a language model with persistent memory, file system access, tool invocation, and network connectivity — broke that boundary. CSA's April 2026 research note "Promptware: When Prompt Injection Becomes C2" documented how researchers formalized the term "promptware" to describe prompt-based payloads sophisticated enough to traverse a full intrusion kill chain: initial access, persistence, lateral movement, and command-and-control, all executed through text rather than executable code [8]. Bruce Schneier and his co-authors, writing about the same underlying academic paper, described this as a seven-stage kill chain that reframes prompt injection from an isolated input-validation bug into an operational attack methodology comparable in sophistication to traditional malware campaigns [3][4].

What has changed in the months since is less about the kill chain itself and more about how the defensive community talks about the artifact that travels through it. Detection engineers writing in July 2026 have begun using the term "semantic malware" to name the payload class directly: malicious logic embedded not in a binary or a script, but in the unstructured text that AI agents are designed to read and act on — configuration files, memory files, tool descriptions, and other artifacts that were never designed to be treated as an execution surface [1][2]. The distinction matters because it reframes the defensive question. A kill-chain framework tells a security team how an attack might progress once it starts; a malware classification tells a security operations center what to build detections for, what to log, and what a signature or behavioral rule should look for. That second question — the detection engineering question — is where the current gap lies, and it is the focus of this note.

Security Analysis

Brainworm: A Working Case Study in Semantic Malware

Origin's Brainworm research, published in March 2026, is among the most detailed public demonstrations of semantic malware as a distinct artifact rather than a theoretical category. The attack targets the memory files that AI coding agents — Claude Code, OpenAI Codex, and Google's Gemini CLI were all named — automatically load at startup to persist context and preferences across sessions, such as CLAUDE.md and AGENTS.md [5]. An attacker who can plant a specification inside one of these trusted files does not need to write any code at all. Instead, the specification describes, in natural language, the behavior of a command-and-control agent: register with a controller, poll for tasking,

execute it using the agent's own tools, and report results back. Brainworm's researchers call this technique Spec-Driven Development, borrowing the name of a legitimate coding-agent workflow pattern to describe an attack that abuses it [5].

The command-and-control channel itself illustrates why this artifact resists conventional analysis. Brainworm registers with a controller over AMQP, the protocol underlying RabbitMQ — an entirely legitimate message-queuing technology with no inherent connection to malware [5]. The researchers note that the same technique could route its tasking channel through GitHub Issues, a shared document, or a chat platform the agent already has credentials to reach, echoing the broader pattern CSA documented in its April note of AI platforms being repurposed as command-and-control infrastructure without any attacker-owned servers [5][8]. What is new in Brainworm is the absence of any artifact at all outside the agent's own context window: no dropped file, no spawned process that persists after the agent's turn ends, nothing for a file scanner or a memory-resident detection agent to inspect once execution completes. Origin's researchers describe the resulting condition plainly: the agent's context window has become a trust domain that requires active defense [5]. That defense is in tension with the automation value organizations deployed these agents to capture in the first place.

Why Endpoint Detection Assumptions Break Down

The deeper problem semantic malware exposes is not that a new attack technique exists, but that the assumptions underlying a generation of endpoint detection logic no longer hold for the systems running AI coding agents. Koifman's analysis of this shift centers on process lineage — the parent-child relationships between running processes that form the backbone of most EDR behavioral detection. An unexpected child process spawned by a document viewer or a browser has long been a reliable signal of compromise, because in ordinary desktop usage that lineage is anomalous [1]. Coding agents invalidate the assumption directly: an agent is designed to spawn arbitrary processes as part of its normal function, and a single task might legitimately chain together curl, git, python, node, grep, and bash in sequence. Koifman observes that the agent's process tree amounts to a superset of nearly every living-off-the-land binary chain an attacker would want, meaning the exact process sequences that detection engineers have spent years building rules to catch are now indistinguishable from routine developer tooling [1].

This is not an argument that endpoint telemetry has become useless. Rather, it relocates the point at which detection has to happen. Standard detection logic remains valid once an agent's actions touch the operating system in a way a non-agentic process would not — writing to a startup folder, modifying a credential store, or opening an outbound connection to an unexpected destination — but it provides no coverage at all for the portion of the attack that occurs purely within the model's context, before any such OS-level action is taken. The detectionengineering.net newsletter's July 2026 coverage of this

same shift frames the practical response as narrowing rather than abandoning existing controls: file integrity monitoring focused specifically on agent configuration and memory files, and application allow-listing that constrains which binaries an agent-driven process tree is permitted to invoke in the first place [2]. Both approaches share a common insight — that if the semantic layer cannot yet be reliably instrumented, the boundary where semantic instructions become concrete actions is still a valid place to build a detection.

The Bridge Back to Conventional Compromise

A separate line of research shows that the distinction between semantic-only attacks and conventional code execution is less durable than it might appear. Microsoft's Defender Security Research Team disclosed in May 2026 that Semantic Kernel, an agent orchestration framework used across LangChain-, CrewAI-, and Semantic Kernel-based deployments, contained vulnerabilities that let a prompt injection payload escalate directly into remote code execution on the host running the agent [6]. One flaw, tracked as CVE-2026-26030, exploited unsafe string interpolation inside a vector-store filter function that used Python's `eval()`, allowing an attacker-controlled prompt to bypass an existing blocklist through Python's class hierarchy and reach arbitrary code execution. A second, CVE-2026-25592, exposed a file-download function to the model as an invokable tool without validating the destination path, letting an attacker write a malicious script directly into a Windows Startup folder for persistence that survives a reboot [6]. Microsoft researchers demonstrated the practical severity of the first flaw by having a single crafted prompt launch a calculator application on the host machine — a proof of concept for what a more consequential payload could instead deliver [6].

These disclosures matter to the semantic malware discussion because they demonstrate the other half of the threat model. Brainworm shows that an attack can accomplish meaningful objectives while remaining entirely inside the context window, generating no artifacts for traditional tooling to find. The Semantic Kernel vulnerabilities show that the same starting point — a prompt injected into an agent's input — can also cross back into the conventional execution layer that EDR was built to monitor, provided the agent framework exposes a function to the model that was never intended to be model-invokable. A defensive posture that treats these as two unrelated problems, addressing only the context-window layer or only the operating-system layer, risks missing whichever half of the threat it did not prioritize. Both disclosed vulnerabilities were patched through framework updates that introduced abstract-syntax-tree allowlists and stricter function-call validation, illustrating that the fix, in this instance, lived in the code that translates model output into system calls rather than in the model itself [6].

Early Detection Engineering Responses

The defensive response to semantic malware is still forming, but a few concrete patterns are emerging from the practitioner community. File integrity monitoring applied specifically to agent memory and configuration files — the CLAUDE.md and AGENTS.md files Brainworm targets — gives defenders a detection point that does not depend on understanding the semantic content of an attack, only on knowing that a file an agent trusts implicitly has changed unexpectedly [2][5]. Application allow-listing, configured to constrain which tools and binaries an agent process is permitted to invoke, narrows the space of actions available to a compromised agent even when the injection itself goes undetected, though it depends on the allow-list itself remaining trustworthy and not becoming the object of a downstream supply-chain compromise. It is plausible that open-source agent frameworks will begin shipping defenses such as memory-file scanning at load time and delimiter-based separation between an agent's own instructions and tool-call results — a direct attempt to restore the instruction-versus-data boundary that promptware exploits — though this note did not identify a publicly documented instance to cite as of July 2026. These are early and partial measures, not a settled architecture, but they mark the beginning of detection engineering treating semantic malware as a first-class category rather than folding it into generic prompt-injection guidance.

Recommendations

Immediate Actions

Security teams operating AI coding agents or other agents with persistent memory should bring the same change-control discipline to agent memory and configuration files that they already apply to source code. File integrity monitoring or version-control diffing on files such as CLAUDE.md, AGENTS.md, and equivalent agent-loaded configuration should treat an unexpected modification as a security event requiring investigation, not routine drift. Organizations running Semantic Kernel or comparable agent orchestration frameworks should confirm they are on patched versions following Microsoft's May 2026 advisory and should independently audit which functions their own integrations expose to model-driven invocation, since Microsoft's disclosure documents this exact anti-pattern — a convenience function exposed to the model without destination or input validation [6] — and there is no reason to expect it is confined to the two CVEs disclosed there. Any agent action that would modify its own persistent memory, startup configuration, or scheduled tasks should require an explicit human approval gate rather than proceeding autonomously.

Short-Term Mitigations

Detection engineering teams should extend their playbooks to log what an agent's context actually contained at the time it took a consequential action, not only the operating-system-level process telemetry that has historically anchored EDR rules; process lineage alone cannot distinguish sanctioned automation from promptware execution once an agent is doing the invoking [1][2]. Network egress from agent runtime environments deserves specific scrutiny for legitimate-looking channels that can double as command-and-control — message queues, issue trackers, and cloud storage buckets the agent is already permitted to reach — since these will not trigger conventional network security controls built to catch attacker-owned infrastructure [5][8]. Where feasible, application allow-listing should be applied to constrain agent-invoked tool and binary execution, understood as a partial control that narrows the blast radius of a successful injection rather than a substitute for addressing the injection itself.

Strategic Considerations

Organizations building AI security programs should treat semantic malware as a formal addition to their malware taxonomy and incident response playbooks, distinct enough from generic "prompt injection" guidance to warrant its own detection signatures, log retention decisions, and response procedures. This note's findings on context-window instrumentation gaps should be read alongside CSA's prior kill-chain research to identify which specific stages — initial access, persistence, command-and-control — currently have the weakest telemetry coverage in a given environment [8]. Finally, the convergence documented here between purely semantic attacks and conventional remote code execution suggests that today's no-binary promptware should be treated as a leading indicator of tomorrow's endpoint compromise rather than a separate, lower-severity category; security teams that build threat-hunting capability now for agent-framework function exposure will be better positioned as more of this bridge is disclosed.

CSA Resource Alignment

This note builds directly on CSA's April 2026 research note, "Promptware: When Prompt Injection Becomes C2", which established the promptware kill chain and documented the first wave of AI-platform-as-C2-infrastructure findings. Readers seeking the full kill-chain taxonomy and CVE landscape should treat that note as the companion reference; this note narrows in specifically on the detection engineering consequences that note's framework implies but does not itself resolve.

CSA's "Large Language Model (LLM) Threats Taxonomy" provides the standardized vocabulary this note relies on for classifying semantic malware as a threat category distinct from the model-manipulation and insecure-supply-chain categories the taxonomy already defines, and organizations formalizing a semantic malware category internally should map it against that existing structure rather than inventing parallel terminology.

CSA's "Securing LLM Backed Systems: Essential Authorization Practices" directly informs the immediate-action recommendation above regarding function exposure audits: its guidance on least-privilege tool access and external authorization checkpoints describes the kind of architectural control that plausibly could have limited the impact of the Semantic Kernel vulnerabilities, particularly CVE-2026-25592's unvalidated file-download destination, independent of whether the underlying framework code was itself patched.

References

[1] Koifman, D. "Detection Engineering in the Era of Semantic Malware." Detect FYI, July 2026.

[2] Detection Engineering Weekly. "DEW #163 - Semantic Malware Detections, Microsoft's CTI REALM evals and Thrunting for Knowledge." detectionengineering.net, July 2026.

[3] Brodt, O., Feldman, E., Schneier, B., Nassi, B. "The Promptware Kill Chain: How Prompt Injections Gradually Evolved Into a Multistep Malware Delivery Mechanism." arXiv:2601.09625, January 2026.

[4] Schneier, B. "The Promptware Kill Chain." Schneier on Security, February 2026.

[5] Turner, M. "Brainworm: Hiding in Your Context Window." Origin Research, March 2026.

[6] Microsoft Defender Security Research Team. "When Prompts Become Shells: RCE Vulnerabilities in AI Agent Frameworks." Microsoft Security Blog, May 2026.

[7] OWASP Foundation. "OWASP Top 10 for Large Language Model Applications." OWASP.

[8] Cloud Security Alliance AI Safety Initiative. "Promptware: When Prompt Injection Becomes C2." CSA AI Safety Initiative, April 2026.

