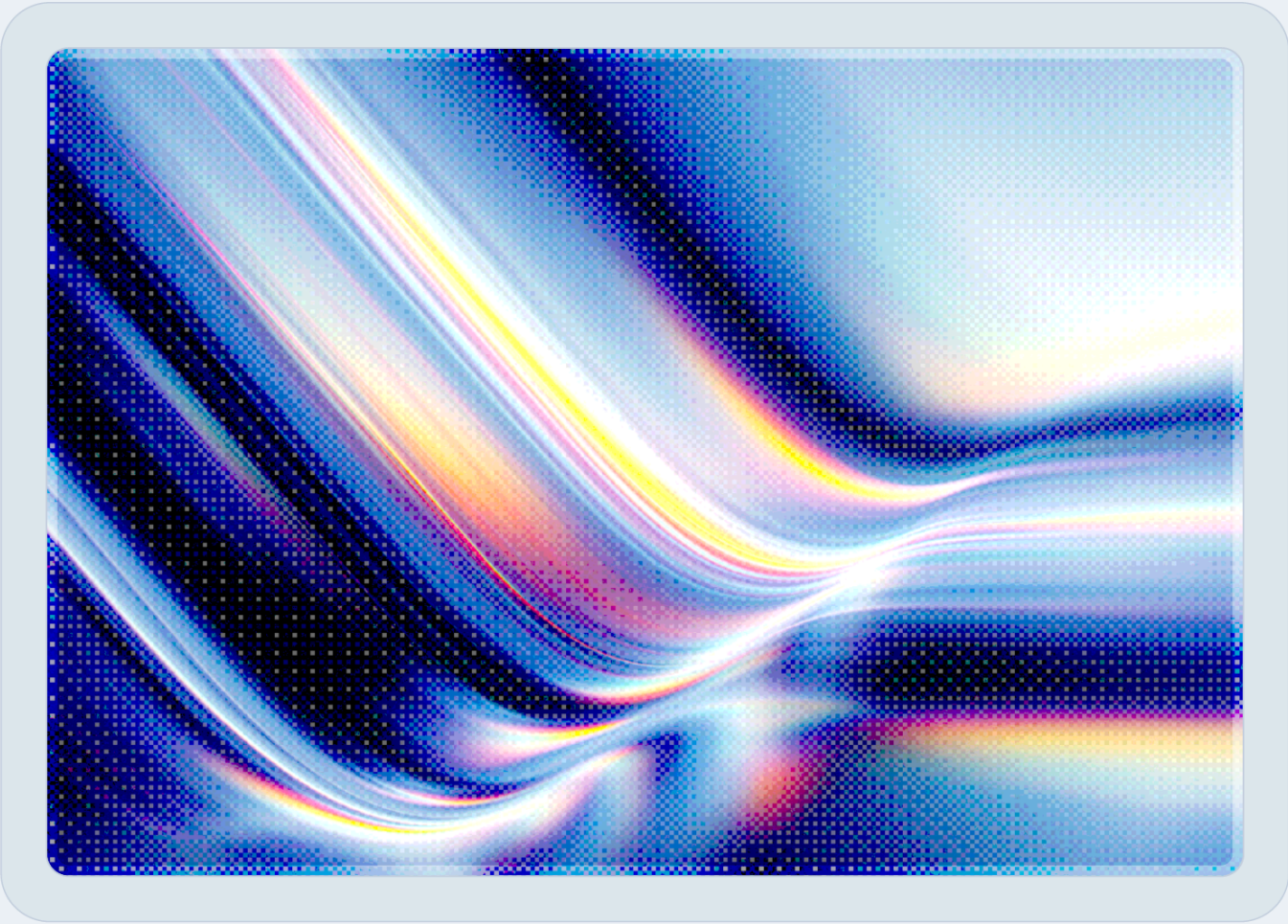
CSAI Foundation | Cloud Security Alliance

DPRK PromptMink: AI-Optimized npm Malware Targeting LLM Agents

2026-05-02

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- North Korean threat actor Famous Chollima has operationalized a novel technique — LLM Optimization (LLMO) abuse — that weaponizes AI coding agents as unwitting malware installers by crafting npm packages whose documentation is engineered to appear ideal to large language models evaluating dependencies.
- The PromptMink campaign, first detected by ReversingLabs and linked to a February 28, 2026 commit to the `openpaw-graveyard` crypto trading agent, demonstrates that AI-generated code commits can introduce malicious dependencies without human review triggering any obvious alarm.
- PromptMink uses a two-layer bait-and-payload architecture: a legitimate-appearing "bait" package silently pulls in a malicious "payload" package as a transitive dependency, bypassing many automated security scans that focus on direct dependencies.
- The malware has evolved from a simple JavaScript infostealer (first uploaded October 2025) through large single-executable bundles to compiled Rust payloads, with more than 300 package versions deployed across the campaign, indicating sustained investment by the threat actor [1].
- Capabilities now include exfiltration of environment files, cryptocurrency wallet credentials, system information, and entire project source trees, plus persistent SSH key injection for remote access on both Linux and Windows [1].
- Organizations using AI coding agents — including Claude Code, GitHub Copilot, and similar tools — must treat agent-authored dependency additions with the same scrutiny applied to human-authored pull requests.

Background

The Democratic People's Republic of Korea (DPRK) has spent more than a decade — since at least 2014 with the Sony Pictures intrusion — building a sophisticated cyber ecosystem that funds its nuclear and missile programs through cryptocurrency theft and financial fraud [9]. What began as opportunistic intrusions against financial institutions has matured into a multi-pronged offensive capability that now includes dedicated units targeting software developers, open-source ecosystems, and the emerging AI

toolchain. The U.S. Treasury Department and CISA have repeatedly attributed these operations to clusters operating under various names — including Lazarus Group, TraderTraitor, and the cluster now designated Famous Chollima, responsible for the Contagious Interview campaign and also tracked under the aliases Shifty Corsair, UNC5342, Tenacious Pungsan [3][4].

Famous Chollima built its reputation on the Contagious Interview campaign, which lured software developers with fake job offers and then delivered malware through a coding challenge GitHub repository. That campaign established the template that PromptMink extends: target developers with something plausible within their professional workflow, hide the payload several layers deep in a dependency chain, and collect credentials or source code that translate into financial gain. The graphalgo campaign, documented by ReversingLabs in early 2026, represented a refinement of this approach using fake recruiting firms on LinkedIn and other developer-adjacent platforms [5].

PromptMink marks a qualitative shift in this tradecraft by targeting the AI coding agent as the vector rather than the developer directly. Rather than relying on a developer to manually clone a repository and run untrusted code, PromptMink is designed so that an AI coding agent — operating autonomously on behalf of a developer — performs the malicious installation. The campaign exploits the fundamental way LLMs evaluate and select software packages: by reading documentation, README files, and package metadata. By crafting packages whose prose is optimized to appeal to LLMs assessing fitness for a project, Famous Chollima has extended the attack surface of software supply chain compromise to include the AI agent itself as an attack vector, not merely a potential target.

Security Analysis

The LLMO Abuse Technique

ReversingLabs coined the term "LLM Optimization (LLMO) abuse" to describe the technique at the heart of PromptMink [1]. Traditional search engine optimization involves crafting text to rank well in algorithmic keyword matching; LLMO abuse involves crafting package documentation so that an LLM evaluating the package — whether recommending it to a developer or adding it as a dependency autonomously — rates it as safe, relevant, and well-maintained. The PromptMink packages contain detailed and professionally written README files, plausible semantic versioning histories, and package names that mimic established utility libraries. When an LLM coding agent working on a cryptocurrency or Solana-adjacent project consults package metadata to evaluate whether `@solana-launchpad/sdk` would satisfy a validation requirement, the documentation is crafted to present the signals a model typically associates with legitimate software [1].

This technique exploits a structural gap in how LLM agents evaluate trust. Human developers can cross-reference package download counts, GitHub star counts, community activity, and contributor reputation. An AI agent without explicit tooling to query npm's registry metadata or cross-reference against known-malicious package databases is left to evaluate documentation alone — a surface the attacker entirely controls.

Two-Layer Bait-and-Payload Architecture

PromptMink's infrastructure relies on structural separation between visibility and execution — a design that appears intended to defeat the security tooling organizations most commonly deploy [1][6]. The "bait" package — in the confirmed February 2026 incident, `@solana-launchpad/sdk` — contains no malicious code whatsoever. It passes static analysis tools, malware scanners, and manual code review without triggering alerts. Its only operative function is to list `@validate-sdk/v2` as a dependency. That payload package, which presents itself as a routine utility SDK for hashing, validation, encoding/decoding, and secure random generation, contains the malicious capability [1].

This architecture is significant for several reasons. First, it means that security teams auditing direct dependencies in a lock file or manifest may completely miss the threat. Second, it means the bait package accumulates a clean history — no flags, no removal notices, no community advisories — while the payload package can be swapped, updated, or rotated without touching the bait layer. Third, it means that an AI agent adding the bait package to a project has plausible deniability: the package it added was, strictly speaking, clean.

ReversingLabs' analysis of the February 28, 2026 incident confirmed that Anthropic's Claude co-authored a commit to the `openpaw-graveyard` autonomous crypto trading agent that introduced `@solana-launchpad/sdk` as a dependency [1]. The commit appeared syntactically normal and was consistent with the project's existing code style. There is no indication that Claude was compromised or manipulated by prompt injection in this specific incident; rather, the LLMO-optimized documentation appears to have caused the model to evaluate the package favorably when it was selected as part of autonomous code generation.

Malware Capabilities and Evolution

The PromptMink payload has evolved substantially since its initial appearance on npm in October 2025 [1][7]. In its earliest form, `@validate-sdk/v2` was a straightforward JavaScript infostealer targeting environment variables and configuration files. By early 2026, the malware had expanded into large single-executable bundles with broader reconnaissance capabilities. The most recent variants use compiled Rust binaries, a deliberate architectural choice that provides two tactical advantages: Rust

binaries lack the human-readable source code that JavaScript packages expose, substantially raising the cost of static analysis, and Rust's cross-platform compilation enables a single payload to operate on both Linux and Windows developer workstations without modification [1][8].

When installed, current PromptMink variants perform a staged exfiltration sequence. The malware first scans for `.env` files and `.json` configuration files that commonly contain API keys, database credentials, and authentication tokens. It then enumerates cryptocurrency wallet configuration files, targeting common formats associated with Solana, Ethereum, and Bitcoin tooling. If running in a sufficiently permissive environment, it compresses and exfiltrates the entire project source tree — a capability that converts a credential-theft incident into full intellectual property compromise. Finally, the malware injects attacker-controlled SSH public keys into authorized_keys files on both Linux and Windows, establishing persistent remote access that survives package removal and credential rotation [1][6].

The campaign has deployed more than 300 distinct package versions to npm, indicating a sustained operational tempo and an infrastructure built for scale [1]. ReversingLabs has observed new package variants appearing in apparent response to detection and takedown events, suggesting that the campaign operators monitor package-scanning feeds and rotate infrastructure accordingly.

Connection to the Broader DPRK Developer-Targeting Ecosystem

PromptMink does not operate in isolation. It is best understood as the latest iteration of Famous Chollima's developer-targeting playbook, which also encompasses the Contagious Interview campaign and the graphalgo fake-recruiter campaign [5]. The graphalgo campaign uses fabricated software companies and fake job interviews to deliver malicious coding assessments; those assessments contain GitHub-hosted projects that, when run, trigger malware installation through a dependency chain similar to PromptMink's two-layer structure.

This portfolio of techniques reflects what appears to be a deliberate operational design: Famous Chollima has multiple pathways into developer environments, and the pathways are mutually reinforcing. A developer who declines a suspicious job interview might still have their AI coding agent silently install a malicious dependency; a developer whose workstation escapes the graphalgo campaign might still have credentials harvested through a PromptMink-infected package. The group has demonstrated willingness to invest in tradecraft that adapts to developer workflows, and the shift to targeting AI coding agents represents the next logical evolution of that strategy.

North Korean actors broadly accounted for 76% of reported cryptocurrency theft losses in 2026, with theft since 2017 estimated to have surpassed $6 billion, according to TRM Labs [2]. These financial incentives drive sustained investment in exactly the kind of developer-targeting supply chain attack that

PromptMink represents.

Recommendations

Immediate Actions

Security teams supporting organizations that use AI coding agents should act on the following controls without delay.

Organizations should configure their AI coding agents to require human approval before any new package dependency is added to a project manifest or lock file. Some commercial AI coding environments offer configuration options for requiring confirmation on file writes; organizations should verify whether their chosen platform provides this control, and advocate for it where it is absent. An agent-authored commit that adds a net-new package should be treated as a pull request requiring review, not a trusted autonomous action.

Security teams should audit all projects where AI coding agents have contributed commits over the past six months for net-new npm dependencies introduced by agent-authored commits. For each such dependency, they should trace the full transitive dependency chain and verify that all packages in that chain appear in a trusted, curated registry or have been independently vetted. Particular attention should be paid to packages in the `@solana-launchpad`, `@validate-sdk`, and similar namespaces identified in PromptMink indicators of compromise [1].

Organizations should immediately enforce automated transitive dependency scanning in CI/CD pipelines using tools capable of resolving and analyzing the full dependency graph, not only direct dependencies. Scanners that analyze only declared dependencies in `package.json` will structurally miss the two-layer PromptMink architecture.

Short-Term Mitigations

Over the next thirty to ninety days, security teams should develop and enforce policies governing the trust model for AI agent actions. An AI coding agent should operate under a principle of least privilege: it should have access only to the files and APIs required for its current task, it should not have outbound network access to arbitrary package registries without review, and it should not have the ability to modify CI/CD pipeline configuration or infrastructure-as-code without human approval.

Teams should evaluate deploying a software composition analysis (SCA) tool that integrates with the LLM agent's toolchain rather than operating solely as a post-commit gate. Vendors have begun offering registry-query tools that coding agents can invoke to verify package provenance before adding a dependency; when an agent cannot verify provenance, it should surface that uncertainty to the developer rather than proceeding. Similarly, organizations should evaluate package allowlisting — particularly for high-risk project types such as cryptocurrency tools and applications handling financial credentials — where only packages on a curated approved list may be added without escalated review.

Developer awareness programs should include explicit guidance on AI agent supply chain risk, as the mechanism — a silently installed transitive dependency — differs materially from the more-familiar risk of running untrusted code from a job-interview repository. Developers who have internalized the lessons of the Contagious Interview campaign may not yet recognize that their AI coding agent can introduce equivalent risk through a package addition that appears stylistically indistinguishable from agent-generated utility code.

Strategic Considerations

The emergence of LLMO abuse as a technique signals that AI coding agents are likely to be a durable target for sophisticated threat actors as agent autonomy and deployment scale continue to grow. Organizations should incorporate AI agent behavior into their threat models and review that threat model on a cadence that keeps pace with agent capability evolution — at minimum annually, but preferably when major new agentic capabilities are deployed.

Software Bill of Materials (SBOM) generation should be extended to capture not only the declared dependency graph but also the provenance of who or what authored each dependency addition — human developer, CI automation, or AI agent. This "AI lineage" metadata will become increasingly important for incident response as AI-authored commits grow as a proportion of organizational codebases.

Standards bodies and registries should consider whether LLMO abuse warrants new detection mechanisms at the package registry layer — for example, anomaly detection on documentation that is unusually well-crafted relative to package age, download history, and contributor activity. The npm registry's existing abuse detection appears calibrated for human-facing social engineering patterns; it has not demonstrated capability to detect documentation engineered to deceive LLMs.

CSA Resource Alignment

The PromptMink campaign implicates multiple layers of CSA's agentic and cloud security frameworks, and organizations should use those frameworks as a structured basis for response planning.

CSA's MAESTRO framework (Multi-Agent Environment, Security, Threat, Risk and Outcome) provides the most directly applicable threat model for this campaign [10]. MAESTRO's seven-layer architecture identifies the Agent Ecosystem layer — which encompasses tool and package dependencies — as a distinct attack surface. PromptMink operates at precisely this layer: the malicious package is not a compromise of the LLM itself but of the external artifact the agent retrieves and integrates. MAESTRO's guidance on supply chain integrity and tool validation controls maps directly to the mitigations described in this note.

The AI Controls Matrix (AICM), CSA's comprehensive AI security control framework, addresses AI supply chain security as a dedicated control domain [11]. The AICM's shared security responsibility model is particularly relevant here: it establishes that while model providers bear responsibility for model behavior, application providers — organizations deploying AI coding agents — bear responsibility for the environment in which those agents operate, including the package registries they query and the permissions they hold.

CSA's Zero Trust guidance applies to AI agent identity and authorization in ways directly relevant to PromptMink. A zero trust posture requires that every action an agent takes — including a dependency addition — be evaluated against a policy that considers the identity of the requester, the sensitivity of the action, and the current context [12]. AI coding agents should not be granted implicit trust simply because they are internal tools; their actions should be subject to the same least-privilege and continuous verification principles applied to human developers.

Organizations engaged with the CSA STAR program should consider whether their AI development pipeline security controls are reflected in their STAR assessments. As AI coding agents become standard development tooling, the security of the agentic development pipeline becomes a material aspect of an organization's overall security posture, warranting inclusion in third-party risk assessments and vendor due diligence questionnaires.

References

[1] ReversingLabs. "Claude adds PromptMink malicious dependency to crypto agent." ReversingLabs Blog, 2026.

[2] TRM Labs. "North Korea accounts for 76% of 2026 crypto hack losses, with theft since 2017 topping $6 billion." The Block, 2026.

[3] CISA. "TraderTraitor: North Korean State-Sponsored APT Targets Blockchain Companies." CISA Cybersecurity Advisory AA22-108A, April 2022.

[4] Threat Intel Report. "FAMOUS CHOLLIMA: DPRK employment fraud and developer-lure intrusion set." TIR, February 2026.

[5] ReversingLabs. "Graphalgo fake recruiter test campaign respawned." ReversingLabs Blog, 2026.

[6] The Hacker News. "New Wave of DPRK Attacks Uses AI-Inserted npm Malware, Fake Firms, and RATs." The Hacker News, April 2026.

[7] Infosecurity Magazine. "Malicious npm Dependency Linked to AI-Assisted Commit Targets Crypto Wallets." Infosecurity Magazine, 2026.

[8] CyberSecurityNews. "Claude-Generated Commit Adds PromptMink Malware to Crypto Trading Agent." CyberSecurityNews, 2026.

[9] CISA. "North Korea Cyber Group Conducts Global Espionage Campaign to Advance Regime's Military and Nuclear Programs." CISA Cybersecurity Advisory AA24-207A, July 2024.

[10] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 2025.

[11] Cloud Security Alliance. "AI Controls Matrix (AICM)." Cloud Security Alliance, 2025.

[12] Cloud Security Alliance. "Applying MAESTRO to Real-World Agentic AI Threat Models: From Framework to CI/CD Pipeline." CSA Blog, February 2026.