INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-23 19:02 UTC

Cross-Ecosystem Supply Chain Attack Hides Linux Malware in PHP Packagist Packages via npm postinstall Hooks

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0358
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	Packagist PHP package registry; Composer dependency manager; GitHub Releases and Actions; confirmed packages: moritz-sauer-13/silverstripe-cms-theme, crosiersource/crosierlib-base, devdojo/wave, devdojo/genesis, katanaui/katana, elitedevsquad/sidecar-laravel, r2luna/brain, baskarcm/tzi-chat-ui; 777+ GitHub repositories referencing same payload
Published	2026-05-23T12:07:51
Discovery Source	Rss

Executive Summary

Attackers injected malicious code into eight PHP packages on the Packagist registry, exploiting a cross-ecosystem gap where PHP projects also run npm for frontend tooling. When any developer or automated build pipeline runs 'npm install', the malicious hook silently downloads and executes a Linux binary on the build host, before any application-level or build-time controls can intervene. Organizations using the affected packages in development or CI/CD pipelines are at risk of compromised build infrastructure, credential theft from build environments, and potential propagation into production deployments.

Technical Analysis

Eight PHP packages hosted on Packagist were backdoored by injecting malicious postinstall hooks into package.json files, not composer.json, exploiting the common pattern of PHP projects using npm for frontend asset compilation. When npm install executes, the postinstall hook fetches a Linux ELF binary from an attacker-controlled GitHub Releases endpoint (attributed to account parikhpreyash4, since removed by GitHub) and executes it at installation time, achieving RCE before any application-layer or runtime control can act. Socket Research confirmed 777+ GitHub repositories reference the same payload. Evidence of GitHub Actions

workflow injection suggests the campaign targets propagation through CI/CD pipelines. Confirmed affected packages: moritz-sauer-13/silverstripe-cms-theme, crosiersource/crosierlib-base, devdojo/wave, devdojo/genesis, katanaui/katana, elitedevsquad/sidecar-laravel, r2luna/brain, baskarcm/tzi-chat-ui. No CVE assigned. Relevant CWEs: CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-494 (Download of Code Without Integrity Check), CWE-693 (Protection Mechanism Failure). MITRE ATT&CK: T1195.001 (Supply Chain Compromise: Compromise Software Dependencies), T1105 (Ingress Tool Transfer), T1546 (Event Triggered Execution), T1059.004 (Command and Scripting Interpreter: Unix Shell), T1072 (Software Deployment Tools), T1027.005 (Indicator Removal from Tools). Patch status: no vendor patch; mitigation is package removal and environment audit. CVSS base estimated at 7.5 (no official vector published).

Action Checklist

1. Step 1: Containment. Immediately audit all Composer and npm dependency manifests across development, CI/CD, and staging environments for the eight confirmed malicious packages: moritz-sauer-13/silverstripe-cms-theme, crosiersource/crosierlib-base, devdojo/wave, devdojo/genesis, katanaui/katana, elitedevsquad/sidecar-laravel, r2luna/brain, baskarcm/tzi-chat-ui. Block outbound connections from build hosts to GitHub Releases endpoints not explicitly whitelisted (NIST SC-7(5), CIS 4.4). Isolate any build host or CI/CD runner that executed npm install against these packages since the packages' last known-good version.
2. Step 2: Detection. Search CI/CD pipeline logs, npm install stdout/stderr, and shell execution logs for postinstall hook activity referencing parikhpreyash4 or unexpected ELF binary downloads. Query endpoint and host logs on Linux build systems for new ELF binary creation events in /tmp or home directories (NIST AU-6, CIS 8.2). Hunt for outbound HTTP/HTTPS GET requests to github.com/releases paths from build hosts at install time. Review GitHub Actions workflow files (.github/workflows/) in all 777+ repositories referencing the payload hash for injected steps (NIST AU-12, NIST SI-4). Use system file analysis to identify unexpected executables placed by npm postinstall processes.
3. Step 3: Eradication. Remove all eight affected packages from composer.json and package.json manifests and lock files. Purge the npm and Composer caches on all affected build hosts (npm cache clean --force; composer clear-cache). Delete any ELF binaries dropped by the postinstall hook; audit /tmp, build workspace directories, and CI/CD runner home directories (NIST SI-3, CIS 2.3). Rotate all credentials, tokens, and secrets present on any build host that executed the malicious hook, including GitHub Actions secrets, cloud provider keys, and registry tokens (NIST AC-2, NIST AC-3). Revoke and reissue any deploy keys associated with affected CI/CD runners.
4. Step 4: Recovery. Rebuild all affected CI/CD runner images from a known-good baseline before restoring pipeline operations (NIST CM-2, CIS 4.6). Re-run full dependency installation on clean, isolated hosts and verify no postinstall hook spawns unexpected child processes. Confirm outbound connections from build hosts are restricted to an approved allowlist (NIST AC-4, CIS 4.4). Enable npm postinstall hook logging and alerting before resuming normal operations. Validate GitHub Actions workflow files in all repositories for unauthorized modifications. Monitor build host process trees for anomalous child processes for a minimum of 30 days post-remediation (NIST SI-4).
5. Step 5: Post-Incident. Implement software composition analysis (SCA) tooling in CI/CD pipelines to detect cross-ecosystem dependency anomalies before installation (NIST SA-12, CIS 7.1). Enforce a policy requiring review of postinstall hooks in package.json for all npm packages in PHP-adjacent projects (CIS 2.1, CIS 2.3). Adopt a dependency pinning strategy using lock files and integrity hashes, validated against a private registry mirror (CWE-494 mitigation, CIS 7.3). Establish a process for monitoring Packagist and

npm advisory feeds for packages in use. Review CI/CD pipeline permissions to enforce least privilege on runners; runners should not have write access to source repositories or production secrets (NIST AC-6, CIS 5.4).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and potentially regulatory notification if forensic analysis confirms the dropped ELF binary exfiltrated environment variables, GitHub Actions secrets, or cloud provider credentials from any runner with access to production systems or data classified as PII, PHI, or subject to PCI-DSS scope, as credential compromise in those contexts may trigger mandatory breach notification obligations.
Recovery Notes	Rebuild all affected CI/CD runners from a pinned, cryptographically verified baseline image rather than restoring from snapshot, as the ELF binary's full capability (persistence mechanisms, lateral movement) may not yet be fully characterized. Monitor all rebuilt runners for 30 days post-remediation using auditd or Sysmon-for-Linux, specifically watching for execve calls from node or npm parent processes, outbound connections from build processes to GitHub Releases CDN endpoints (objects.githubusercontent.com), and any new ELF files created in /tmp or runner home directories. Validate that all rotated credentials — GitHub Actions secrets, cloud provider keys, deploy keys, and registry tokens — have been confirmed revoked by querying each provider's audit log to ensure the old credentials produced no successful authentications after the rotation timestamp.
Forensic Artifacts	ELF binary dropped by npm postinstall hook: located in /tmp, ~/build workspace, or CI/CD runner home directory; SHA-256 hash must be captured before deletion; binary analysis via 'strings', 'readelf -a', and sandbox execution (e.g., in a Cuckoo or any-run sandbox) will reveal C2 infrastructure, exfiltration targets, and persistence mechanisms specific to this campaign | npm install execution logs: ~/.npm/_logs/*.log and CI/CD pipeline stdout/stderr capturing the exact postinstall hook command string, download URL referencing github.com/parikhpreyash4 or GitHub Releases CDN, and the binary filename — this is the primary evidence tying the infection to the specific malicious Packagist/npm package | Runner environment variable dump (printenv output): captures all secrets, tokens, and credentials in the runner process environment at time of postinstall hook execution; this defines the credential compromise blast radius and is the highest-priority forensic artifact for determining breach notification obligations | auditd or Sysmon-for-Linux process execution logs: EventID 11 (FileCreate) and EventID 3 (NetworkConnect) entries showing the process lineage npm → sh/bash → curl/wget → chmod +x → execve of the downloaded ELF, providing the kill chain evidence specific to the postinstall hook execution mechanism | GitHub Actions workflow run logs and secrets audit log: the workflow run log captures which npm install step triggered the postinstall hook and which runner executed it; the repository's Security log (Settings > Security > Audit log) records any GitHub Actions secret reads during the malicious run, establishing whether repository-scoped secrets were exposed to the attacker

Per-Action IR Details

Step 1: Containment — Immediately audit all Composer and npm dependency manifests across development, CI/CD, and staging environments for the eight confirmed malicious packages: moritz-sauer-13/silverstripe-cms-theme, crosiersource/crosierlib-base, devdojo/wave, devdojo/genesis, katanaui/katana, elitedevsquad/sidecar-laravel, r2luna/brain, baskarcm/tzi-chat-ui. Block outbound connections from build hosts to GitHub Releases endpoints not explicitly whitelisted (NIST SC-7, CIS 4.4).

Isolate any build host or CI/CD runner that executed npm install against these packages since the packages' last known-good version.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST SC-7 (Boundary Protection), NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 12.2 (Establish and Maintain a Secure Network Architecture)

Compensating: Run 'grep -rE "moritz-sauer-13|crosiersource|devdojo/wave|devdojo/genesis|katanaui|elitedevsquad|r2luna/brain|baskarcm" composer.json composer.lock package.json package-lock.json' across all project roots. Block outbound TCP 443 from CI/CD runner hosts to github.com/parikhpreyash4 and github.com/releases paths using iptables: 'iptables -I OUTPUT -p tcp --dport 443 -d 140.82.112.0/20 -j DROP' (adjust for your GitHub IP range from meta.github.com). Snapshot the runner's filesystem with 'dd' or 'rsync --archive' before isolation to preserve forensic state.

Evidence: Before isolating the runner, capture: (1) full output of 'npm install' run logs from CI/CD pipeline stdout/stderr showing postinstall hook execution and any download URLs referencing github.com/parikhpreyash4 or similar; (2) filesystem listing of /tmp, ~/build, and the npm cache directory (~/.npm/_cacache) timestamped to the install window; (3) network connection state via 'ss -tunap' or 'netstat -tunap' and any active HTTPS sessions from the runner to GitHub Releases CDN endpoints; (4) running process list ('ps auxf') to identify any ELF binary still resident in memory spawned by the postinstall hook.

Step 2: Detection — Search CI/CD pipeline logs, npm install stdout/stderr, and shell execution logs for postinstall hook activity referencing parikhpreyash4 or unexpected ELF binary downloads. Query endpoint and host logs on Linux build systems for new ELF binary creation events in /tmp or home directories (NIST AU-6, CIS 8.2). Hunt for outbound HTTP/HTTPS GET requests to github.com/releases paths from build hosts at install time. Review GitHub Actions workflow files (.github/workflows/) in all 777+ repositories referencing the payload hash for injected steps (NIST AU-12, NIST SI-4). Use D3-SFA (System File Analysis) to identify unexpected executables placed by npm postinstall processes.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs), CIS 8.11 (Conduct Audit Log Reviews)

Compensating: Deploy Sysmon on any Linux build hosts using sysmon-for-linux; configure EventID 11 (FileCreate) to alert on ELF files created in /tmp or /home by node or npm processes, and EventID 3 (NetworkConnect) to capture outbound connections from 'npm' or 'node' parent processes. Without Sysmon, use auditd rules: 'auditctl -w /tmp -p wa -k elf_drop' and 'auditctl -a always,exit -F arch=b64 -S execve -k exec_track'. Search pipeline logs with: 'grep -E "parikhpreyash4|github.com/releases|postinstall" ~/.npm/_logs/*.log /var/log/gitlab-runner/*.log'. Use 'find /tmp /home -type f -newer /usr/bin/npm -executable -exec file {} \; | grep ELF' to locate dropped binaries. For GitHub Actions workflow scanning, run: 'grep -rE "parikhpreyash4|curl.*github.com/releases|wget.*ELF" .github/workflows/'.

Evidence: Capture before analysis: (1) npm debug logs at ~/.npm/_logs/ and any CI/CD runner log artifacts showing the exact postinstall hook command string executed by the malicious package.json scripts field; (2) auditd or Sysmon logs showing process lineage — specifically node/npm spawning sh/bash which spawned curl or wget to a github.com/releases URL, then chmod +x and execve of the downloaded ELF; (3) full 'strace -f -e trace=execve,openat,connect' output if the binary can be safely re-executed in an isolated sandbox; (4) GitHub Actions run logs for all workflows in the 777+ flagged repositories, filtering for steps that invoke 'npm install' or 'composer install' against the eight named packages; (5) DNS query logs from the build network for resolution of github.com subdomains or CDN endpoints (objects.githubusercontent.com) during the install window.

Step 3: Eradication — Remove all eight affected packages from composer.json and package.json manifests and lock files. Purge the npm and Composer caches on all affected build hosts (npm cache clean --force; composer clear-cache). Delete any ELF binaries dropped by the postinstall hook; audit /tmp, build workspace directories, and CI/CD runner home directories (NIST SI-3, CIS 2.3). Rotate all credentials, tokens, and secrets

present on any build host that executed the malicious hook, including GitHub Actions secrets, cloud provider keys, and registry tokens (D3-CRO, NIST AC-2). Revoke and reissue any deploy keys associated with affected CI/CD runners.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-3 (Malicious Code Protection), NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), NIST CM-6 (Configuration Settings), CIS 2.3 (Address Unauthorized Software), CIS 5.3 (Disable Dormant Accounts)

Compensating: Automate cache and binary removal: 'npm cache clean --force && composer clear-cache && find /tmp /home /root /var/lib/gitlab-runner -type f -executable | xargs file | grep ELF | cut -d: -f1 | xargs rm -fv'. For credential rotation without a secrets manager, enumerate exposed secrets by scanning runner environment variables: 'printenv | grep -iE "token|secret|key|password|aws|gcp|azure"' and cross-reference against GitHub Actions secrets configured in affected repositories via the GitHub API ('gh secret list --repo /'). Revoke GitHub personal access tokens and deploy keys via GitHub UI under Settings > Developer Settings and Settings > Deploy Keys. For cloud keys, use AWS CLI 'aws iam delete-access-key' or equivalent GCP/Azure commands. Use ClamAV with a custom YARA rule matching the known ELF binary hash (if available from threat intel) to scan all runner workspaces: 'clamscan -r --include="*" /tmp /home /var/lib/runner'.

Evidence: Before purging, preserve forensic copies: (1) hash (SHA-256) and binary copy of any ELF executable found in /tmp or build directories — this is the primary malware artifact from the postinstall hook and must be retained for malware analysis; (2) full environment variable dump from the compromised runner at time of infection ('printenv > /forensics/runner_env_$(date +%s).txt') to establish exactly which secrets were in scope; (3) Git history and diff of composer.json, composer.lock, package.json, and package-lock.json showing when the malicious package was introduced and by whom ('git log -p -- composer.json package.json'); (4) GitHub Actions secrets audit log from the repository's Settings > Security log to determine if any secret was read or exported during the malicious run.

Step 4: Recovery — Rebuild all affected CI/CD runner images from a known-good baseline before restoring pipeline operations (NIST CM-2, CIS 4.6). Re-run full dependency installation on clean, isolated hosts and verify no postinstall hook spawns unexpected child processes. Confirm outbound connections from build hosts are restricted to an approved allowlist (NIST AC-4, CIS 4.4). Enable npm postinstall hook logging and alerting before resuming normal operations. Validate GitHub Actions workflow files in all repositories for unauthorized modifications (D3-SICA). Monitor build host process trees for anomalous child processes for a minimum of 30 days post-remediation (NIST SI-4).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST CM-2 (Baseline Configuration), NIST AC-4 (Information Flow Enforcement), NIST SI-4 (System Monitoring), NIST CP-9 (System Backup), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Rebuild CI/CD runners from a pinned Docker image digest (e.g., 'FROM ubuntu@sha256:') rather than a mutable tag to prevent image drift. Validate clean reinstallation by running 'strace -f -e trace=execve,connect npm install 2>&1 | grep -E "execve|connect"' on the rebuilt host and confirming no connections to github.com/releases or execution of downloaded binaries. Implement npm lifecycle hook visibility: add 'npm config set ignore-scripts false' and pipe 'npm install --foreground-scripts' output to a log file audited by auditd. For GitHub Actions workflow integrity, use 'git log --all --diff-filter=M -- .github/workflows/' to identify any workflow file modified since the last known-good commit. Monitor process trees on recovered runners with a cron-based 'ps auxf >> /var/log/runner_proctree.log' every 5 minutes for 30 days, alerting on node/npm spawning unexpected child processes.

Evidence: Before returning runners to production, document: (1) Docker image digest or VM snapshot hash of the rebuilt runner baseline as the new known-good reference; (2) 'npm install' execution trace from the clean rebuilt host confirming no outbound connections to github.com/parikhpreyash4 or objects.githubusercontent.com during install of the replacement (safe) packages; (3) GitHub Actions workflow file hashes ('sha256sum .github/workflows/*.yml') for all affected repositories, establishing integrity baseline for ongoing monitoring; (4) outbound firewall rule export confirming github.com/releases endpoints are now blocked or restricted to an allowlist, satisfying NIST AC-4 information flow

enforcement for build infrastructure.

Step 5: Post-Incident — Implement software composition analysis (SCA) tooling in CI/CD pipelines to detect cross-ecosystem dependency anomalies before installation (NIST SA-12, CIS 7.1). Enforce a policy requiring review of postinstall hooks in package.json for all npm packages in PHP-adjacent projects (CIS 2.1, CIS 2.3). Adopt a dependency pinning strategy using lock files and integrity hashes, validated against a private registry mirror (CWE-494 mitigation, CIS 7.3). Establish a process for monitoring Packagist and npm advisory feeds for packages in use. Review CI/CD pipeline permissions to enforce least privilege on runners — runners should not have write access to source repositories or production secrets (NIST AC-6, CIS 5.4).

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity

Controls: NIST SA-12 (Supply Chain Protection), NIST AC-6 (Least Privilege), NIST SA-15 (Development Process, Standards, and Tools), NIST SI-2 (Flaw Remediation), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts)

Compensating: For zero-budget SCA, integrate 'npm audit --audit-level=high' and 'composer audit' as mandatory pipeline steps that fail the build on HIGH/CRITICAL findings. To detect cross-ecosystem postinstall hook abuse specifically, add a pre-install script that parses all package.json files in node_modules for 'postinstall' keys containing curl, wget, or exec: 'grep -rE "postinstall.*curl|postinstall.*wget|postinstall.*exec" node_modules/*/package.json'. Subscribe to Packagist security advisories at https://packagist.org/security-advisories and npm security advisories via 'npm audit' RSS or the GitHub Advisory Database. For private registry mirroring at no cost, use Verdaccio (npm) and Satis (Composer) to proxy and pin packages. Enforce runner least privilege by scoping GitHub Actions GITHUB_TOKEN permissions in workflow YAML: 'permissions: contents: read' and removing repository write access from runner service accounts.

Evidence: For the lessons-learned record, compile: (1) timeline reconstruction from CI/CD logs showing the first 'npm install' execution against any of the eight malicious packages, establishing patient-zero build host and infection window; (2) inventory of all GitHub Actions secrets, cloud credentials, and deploy keys that were in scope on affected runners during the infection window — this defines the full credential compromise blast radius for the post-incident report; (3) Packagist and npm registry metadata for each of the eight packages documenting the version where the malicious postinstall hook was introduced vs. the last known-good version, to establish dwell time; (4) list of all 777+ GitHub repositories confirmed to reference the payload hash, sourced from the original threat intelligence, to drive outreach and remediation tracking.

Detection Guidance

Primary detection surface is the build environment, not production. Key indicators: (1) npm postinstall hook execution on any package.json file in a PHP project repository; flag any postinstall script that invokes curl, wget, fetch, or a shell command downloading from github.com/releases or any external URL. (2) ELF binary creation events in build workspace directories, /tmp, or CI/CD runner home directories during or immediately after npm install. (3) Outbound HTTP/HTTPS requests from build hosts to github.com/parikhpreyash4 or any GitHub Releases URL not in your approved artifact allowlist. (4) Unexpected child processes spawned from the Node.js npm process tree, specifically sh, bash, or curl/wget as direct npm children. (5) GitHub Actions workflow modifications adding new steps, jobs, or uses: references not present in your baseline (T1546, T1072). Query SIEM for process creation events where parent process matches node or npm and child process matches curl, wget, sh, or bash on Linux build hosts (NIST AU-6, CIS 8.2). IOC pattern: HTTP GET to github.com/*/releases/download/* from a Node.js or npm process. Cross-reference against the 777+ affected repository list from Socket Research's disclosure. System file analysis and configuration analysis are applicable

countermeasures for ongoing monitoring.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`github.com/parikhpreyash4`	Attacker-controlled GitHub account hosting malicious ELF binary payload via GitHub Releases; account no longer active but artifact URLs may persist in cached pipeline configs	HIGH
URL	`github.com/parikhpreyash4/*/releases/download/*`	Pattern for attacker-hosted ELF binary download URLs referenced in malicious postinstall hooks; block this pattern at egress firewall on build hosts	HIGH
HASH	`[not published in available sources]`	ELF binary hash was not disclosed in T3 sources reviewed; obtain from Socket Research's full disclosure report for hash-based detection	LOW

Framework Mappings

MITRE-ATTACK

- **T1105** — Ingress Tool Transfer
- **T1553** — Subvert Trust Controls
- **T1546** — Event Triggered Execution
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1036.005** — Match Legitimate Resource Name or Location
- **T1204.002** — Malicious File
- **T1072** — Software Deployment Tools
- **T1574** — Hijack Execution Flow
- **T1027.005** — Indicator Removal from Tools
- **T1059.004** — Unix Shell

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control

- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **15.1** — Establish and Maintain an Inventory of Service Providers

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

ISO-27001-2022

- **A.5.21** — Managing information security in the ICT supply chain

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1105	Ingress Tool Transfer	Command-And-Control
T1553	Subvert Trust Controls	Defense-Evasion
T1546	Event Triggered Execution	Privilege-Escalation
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion
T1204.002	Malicious File	Execution
T1072	Software Deployment Tools	Execution
T1574	Hijack Execution Flow	Persistence
T1027.005	Indicator Removal from Tools	Defense-Evasion
T1059.004	Unix Shell	Execution

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/packagist-supply-chain-attack-inf...	T3

Source	URL	Tier
Malicious Postinstall Hook Found Across 700+ GitHub Reposito...	https://socket.dev/blog/malicious-postinstall-hook-found-across-700...	T3
Supply Chain Attack Flags 700+ GitHub Repos W... - CoinStats	https://coinstats.app/news/31b41ef2cf0260c8422744123efbf44b319656d8.. .	T3
International Cyber Digest	https://x.com/IntCyberDigest/status/2057982791917109531/photo/2	T3
Packagist - The PHP Package Repository	https://packagist.org/explore/	T3

Generated 2026-05-23 19:02 UTC by TJS Security Command Center