INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-18 18:46 UTC

Dependency Automation Tools Dependabot and Renovate Abused as Malware Delivery Vectors

THREAT CAMPAIGN | **HIGH** | CVSS 8.1

SCC Item ID	SCC-CAM-2026-0186
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	8.1
Affected Products	GitHub Dependabot, Mend Renovate, all versions used in automated CI/CD and dependency management workflows
Published	2026-04-17
Discovery Source	Gemini

Executive Summary

Attackers are exploiting the automated trust granted to dependency management tools Dependabot and Renovate to inject malicious code into software build pipelines. Any organization using these tools in CI/CD workflows, across all versions and hosting environments, is potentially exposed. A successful attack can deliver malware directly into production software, bypassing traditional code review controls and threatening the integrity of every application built through affected pipelines.

Technical Analysis

This campaign abuses legitimate dependency automation workflows rather than exploiting a software defect in Dependabot or Renovate. No CVE has been assigned. Attackers publish malicious packages to public registries (npm, PyPI, others) or register typosquatted package names that closely resemble legitimate dependencies. Dependabot and Renovate, operating with elevated repository permissions, detect these packages as available updates and automatically open pull requests proposing the malicious dependency. In repositories configured for auto-merge, or where PR reviews are cursory, the malicious package enters the build pipeline and executes during CI/CD runs. CWE coverage: 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 mapping: T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools), T1554 (Compromise Host Software Binary), T1072 (Software Deployment Tools), T1059 (Command and Scripting Interpreter). The attack window between malicious package publication and pipeline execution can be measured in minutes in fully automated environments.

Action Checklist

1. Step 1: Containment, Immediately disable auto-merge on all Dependabot and Renovate pull requests across every repository. In GitHub, navigate to repository Settings > Branches > Branch Protection Rules and disable the 'Allow auto-merge' option. You may also configure this via the GitHub API by setting allow_auto_merge to false. In Renovate, set automerge: false in renovate.json. Audit which repositories currently have auto-merge enabled and treat each as potentially exposed.
2. Step 2: Detection, Review CI/CD pipeline logs for dependency update PRs merged in the last 30 days. Query package manager lock files (package-lock.json, requirements.txt, Pipfile.lock, go.sum) for packages with low download counts, recent publication dates, or names closely resembling established libraries. Cross-reference all bot-proposed packages against OSV.dev (https://osv.dev), npm audit, and PyPI safety databases, understanding that newly published malicious packages may not yet appear in these databases. Manual review of package metadata (author, repository links, recent publication date) is critical for detection of fresh typosquatting attempts.
3. Step 3: Eradication, Enable registry-level integrity checks: npm enforce --audit-level=critical, pip install with hash verification (--require-hashes), and equivalent controls for other package managers in use. Configure Dependabot and Renovate to restrict update proposals to packages above a minimum download threshold and minimum publication age (recommended: 7 days minimum). Add CODEOWNERS rules requiring human review on all dependency update PRs.
4. Step 4: Recovery, Re-examine and re-build any artifact produced during the exposure window using a clean, verified dependency lockfile. Scan built artifacts with a software composition analysis (SCA) tool. Monitor runtime environments for anomalous outbound connections, unexpected process spawning, or new scheduled tasks following recent dependency updates. Treat any pipeline that ran during the exposure window as potentially compromised until artifact integrity is confirmed.
5. Step 5: Post-Incident, Conduct a dependency governance review: document which repositories use automated dependency tools, what permissions those tools hold, and what merge controls exist. Implement a minimum-age policy for all new dependency versions (reject packages published less than 7 days ago). Integrate OSV.dev, Socket.dev, or equivalent supply chain scanning into all CI/CD pipelines. Add this attack pattern to developer security training focused on supply chain risk.

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate to CISO and legal immediately if SCA scanning or runtime behavioral analysis confirms a malicious package was executed in a production environment processing PII, PHI, or financial data — this triggers breach notification obligations under GDPR Article 33, HIPAA §164.410, or state breach notification statutes depending on jurisdiction.

Recovery Notes	All build artifacts produced during the exposure window must be treated as untrusted until rebuilt from a verified, hash-pinned dependency lockfile; do not re-deploy or distribute artifacts from the compromised window under any circumstances. Monitor production systems that ran software built during the exposure window for 30 days post-recovery, specifically watching for scheduled task creation, unexpected outbound DNS/HTTP from application processes, and new local user accounts — behaviors consistent with malicious npm/PyPI package post-install scripts achieving persistence. Re-run SCA scanning against all rebuilt artifacts weekly for the first month to catch any delayed-activation supply chain payloads that may not have triggered during initial analysis.
Forensic Artifacts	GitHub Audit Log (Settings > Security > Audit log or API: /orgs/ORG/audit-log) — filtered for `repo.auto_merge_enabled`, `pull_request.auto_merge_enabled`, and all merge events attributed to `dependabot[bot]` or `renovate[bot]` during the exposure window; this is the primary evidence of which automated merges occurred and under what permissions. | Package manager lock file git history — `git log --all --author='dependabot|renovate' -p -- '**/package-lock.json' '**/requirements.txt' '**/Pipfile.lock' '**/go.sum'` — preserving the exact diff of every dependency added, updated, or removed by bot-authored commits; this is ground truth for identifying which packages were injected. | CI/CD pipeline execution logs tied to bot-merged PRs — GitHub Actions run logs (`/repos/ORG/REPO/actions/runs` API filtered by triggering PR author), Jenkins build logs (`$JENKINS_HOME/jobs//builds//log`), or GitLab CI job traces — capturing the full build environment, installed package versions, and any anomalous install-script output (e.g., base64-encoded commands, unexpected network calls during `npm install` or `pip install`). | npm/PyPI registry metadata for all packages introduced during the exposure window — captured via `npm view --json` or PyPI JSON API (`https://pypi.org/pypi//json`) — preserving publication timestamp, author, download count, and `scripts.postinstall` field; malicious packages in this campaign typically show very recent publication dates, single-version history, and postinstall scripts containing obfuscated execution payloads. | Runtime outbound network connection logs from application servers or containers that executed software built during the exposure window — specifically netflow or firewall logs showing connections from application process PIDs to non-registry external IPs, which would confirm command-and-control callback or data exfiltration behavior consistent with MITRE ATT&CK T1071.001 (Application Layer Protocol: Web Protocols) and T1048 (Exfiltration Over Alternative Protocol).

Per-Action IR Details

Step 1: Containment — Immediately disable auto-merge on all Dependabot and Renovate pull requests across every repository. In GitHub, navigate to repository Settings > Branches and remove auto-merge permissions for bot accounts. In Renovate, set automerge: false in renovate.json. Audit which repositories currently have auto-merge enabled and treat each as potentially exposed.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: Stop ongoing damage by removing the automated trust mechanism that allows malicious dependency PRs to merge without human review.

Controls: NIST IR-4 (Incident Handling), NIST CM-3 (Configuration Change Control), NIST SA-12 (Supply Chain Protection), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Use the GitHub CLI to enumerate all repositories with auto-merge enabled: `gh repo list ORG --limit 1000 --json name,autoMergeAllowed | jq '.[] | select(.autoMergeAllowed==true) | .name'`. For Renovate, run `grep -r 'automerge' .renovaterc* renovate.json* **/renovate.json` across all repository clones to identify any non-false automerge configurations. Maintain a spreadsheet of findings as a chain-of-custody record before making changes.

Evidence: Before disabling auto-merge, capture: (1) GitHub audit log export (Settings > Security > Audit log, filter by `action:repo.auto_merge_enabled`) documenting which bot accounts held auto-merge permissions and when they were granted; (2) Full list of all Dependabot and Renovate PRs merged in the last 30 days via GitHub API: `gh pr list --repo ORG/REPO --state merged --author 'dependabot[bot]' --json number,title,mergedAt,author`; (3) Snapshot of current `renovate.json` and `.github/dependabot.yml` configs in each repository before modification.

Step 2: Detection — Review CI/CD pipeline logs for dependency update PRs merged in the last 30 days. Query package manager lock files (package-lock.json, requirements.txt, Pipfile.lock, go.sum) for packages with low download counts, recent publication dates, or names closely resembling established libraries. Cross-reference all bot-proposed packages against npm audit, PyPI safety DB, or OSV.dev before treating any recent Dependabot or Renovate merge as clean.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: Identify which Dependabot/Renovate-merged dependency changes introduced malicious packages by correlating lock file diffs with known-bad indicators across OSV.dev, npm audit, and PyPI Safety DB.

Controls: NIST SI-3 (Malicious Code Protection), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST RA-5 (Vulnerability Monitoring and Scanning), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Extract all dependency changes introduced by Dependabot/Renovate PRs in the last 30 days using: `git log --all --author='dependabot\|renovate' --since='30 days ago' --diff-filter=A -- '**/package-lock.json' '**/requirements.txt' '**/go.sum' | git diff-tree --stdin -r`. Pipe new package names to OSV.dev batch API: `curl -X POST https://api.osv.dev/v1/querybatch -d @batch_query.json`. For typosquatting detection, run the free `confused` tool or `doppelganger` against each new package name to identify look-alike packages targeting popular libraries (e.g., 'lodash' vs 'l0dash').

Evidence: Capture before analysis: (1) Git diff of lock files (`package-lock.json`, `requirements.txt`, `Pipfile.lock`, `go.sum`) for every Dependabot/Renovate PR merged in the 30-day window — `git show :package-lock.json | diff - package-lock.json`; (2) CI/CD pipeline execution logs from GitHub Actions (`.github/workflows/` run history), GitLab CI (`/var/log/gitlab-runner/`), or Jenkins (`$JENKINS_HOME/jobs//builds/`) for every build triggered by a bot-authored PR; (3) npm registry metadata for each new or updated package: `npm view --json | jq '{name,version,time,downloads}'` — capture publication timestamps and download counts as baseline IOC evidence.

Step 3: Eradication — Enable registry-level integrity checks: npm enforce --audit-level=critical, pip install with hash verification (--require-hashes), and equivalent controls for other package managers in use. Configure Dependabot and Renovate to restrict update proposals to packages above a minimum download threshold and minimum publication age (recommended: 7 days minimum). Add CODEOWNERS rules requiring human review on all dependency update PRs.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: Remove the conditions that allowed malicious packages to enter the pipeline by enforcing cryptographic hash verification and minimum-age policies that prevent zero-day malicious package publication from being auto-ingested.

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), NIST SA-15 (Development Process, Standards, and Tools), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: For hash verification without enterprise tooling: generate and commit a `pip` constraints file with hashes using `pip-compile --generate-hashes requirements.in > requirements.txt`, then enforce with `pip install --require-hashes -r requirements.txt` in all CI pipeline steps. For npm, add `--require-hashes` equivalent via `.npmrc`: set `audit=true` and `package-lock=true`. Implement a free Socket.dev GitHub App (free tier available) to block Dependabot/Renovate PRs that introduce packages with suspicious publication patterns. For CODEOWNERS enforcement without enterprise GitHub, use the free `reviewdog` GitHub Action to require sign-off on any diff touching lock files.

Evidence: Before applying eradication controls, capture: (1) Current state of `.npmrc`, `pip.conf`, and `pyproject.toml` across all repositories to document the pre-remediation security posture; (2) Existing `CODEOWNERS` files (`.github/CODEOWNERS`) to record what review requirements were absent; (3) Dependabot configuration (`.github/dependabot.yml`) and Renovate configuration (`renovate.json`) with exact `automerge`, `schedule`, and `allowedVersions` settings that permitted the attack surface — these are primary evidence of misconfiguration.

Step 4: Recovery — Re-examine and re-build any artifact produced during the exposure window using a clean, verified dependency lockfile. Scan built artifacts with a software composition analysis (SCA) tool. Monitor runtime environments for anomalous outbound connections, unexpected process spawning, or new scheduled tasks following recent dependency updates. Treat any pipeline that ran during the exposure window as potentially compromised until artifact integrity is confirmed.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: Restore pipeline integrity by rebuilding artifacts from a verified dependency state and confirming no malicious package behavior persists in runtime environments that consumed bot-merged dependencies.

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST IR-4 (Incident Handling), NIST AU-12 (Audit Record Generation), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Run free OWASP Dependency-Check (`dependency-check.sh --project --scan ./`) against rebuilt artifacts to confirm no known-malicious packages remain. For runtime behavioral monitoring without EDR, deploy Sysmon with a supply-chain-focused config (SwiftOnSecurity or Olaf Hartong's modular config) and watch for: Event ID 1 (Process Create) with parent processes matching Node.js (`node.exe`), Python (`python.exe`), or build tools spawning unexpected child processes (e.g., `cmd.exe`, `powershell.exe`, `curl`, `wget`). Monitor outbound connections using `netstat -antp` on Linux or `Get-NetTCPConnection` on Windows, filtering for connections from build tool processes to non-registry IPs. Watch for new crontab entries (`/var/spool/cron/`, `/etc/cron.d/`) or Windows scheduled tasks (`schtasks /query /fo LIST`) created after pipeline runs.

Evidence: Before rebuilding, preserve: (1) Exact build artifacts (container images, compiled binaries, npm/PyPI packages) produced during the exposure window — store immutably with SHA-256 hashes for later forensic comparison; (2) Runtime process snapshots from systems that executed software built during the exposure window: `ps auxf` (Linux) or `Get-Process | Select-Object Name,Id,Path,StartTime` (Windows); (3) Outbound network connection logs from the exposure window — specifically connections from CI/CD runner IPs or application servers to external endpoints not matching known npm/PyPI/GitHub registries, which would indicate phone-home behavior from a malicious package (MITRE ATT&CK T1071.001 — Application Layer Protocol); (4) Crontab and scheduled task listings (`crontab -l`, `ls -la /etc/cron.d/`, `schtasks /query`) captured before any cleanup to detect persistence mechanisms dropped by malicious packages (MITRE ATT&CK T1053 — Scheduled Task/Job).

Step 5: Post-Incident — Conduct a dependency governance review: document which repositories use automated dependency tools, what permissions those tools hold, and what merge controls exist. Implement a minimum-age policy for all new dependency versions (reject packages published less than 7 days ago). Integrate OSV.dev, Socket.dev, or equivalent supply chain scanning into all CI/CD pipelines. Add this attack pattern to developer security training focused on supply chain risk.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Document lessons learned specific to automated dependency tool abuse, update the IR plan to include supply chain pipeline compromise as a named incident category, and share IOCs (malicious package names, registry patterns) with peers via ISAC or public threat feeds.

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SA-12 (Supply Chain Protection), NIST PM-30 (Supply Chain Risk Management Strategy), NIST SI-5 (Security Alerts, Advisories, and Directives), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Build a dependency tool inventory using `gh repo list ORG --limit 1000 --json name | xargs -I{} gh api repos/ORG/{}/contents/.github/dependabot.yml` and equivalent Renovate config search to produce a repository-level

risk register. Implement free OpenSSF Scorecard (`scorecard --repo github.com/ORG/REPO`) across all repositories to surface dependency update policy gaps — focus on the 'Dangerous-Workflow' and 'Token-Permissions' checks directly relevant to this attack vector. Add a free OSSF Allstar GitHub App policy enforcing branch protection and CODEOWNERS requirements organization-wide. Document malicious package names and registry metadata patterns as YARA rules targeting `package.json`, `requirements.txt`, and `go.mod` files in future PRs.

Evidence: For the post-incident record, compile: (1) Complete audit log of all Dependabot and Renovate activity for the exposure window, exported from GitHub Audit Log API (`/orgs/ORG/audit-log?phrase=dependabot&include=all`); (2) Final lock file diffs showing exact packages introduced during the exposure window versus the clean rebuilt state — this is the primary artifact for regulatory disclosure if a malicious package reached production; (3) Timeline reconstruction mapping bot PR merge timestamps to CI/CD pipeline execution times to artifact publication times, supporting breach notification decisions if PII/PHI was processed by potentially compromised software.

Detection Guidance

Query version control audit logs for all pull requests opened by Dependabot or Renovate bot accounts over the past 30 days; flag any that were auto-merged without human approval. In GitHub, use the Audit Log API filtering on actor:dependabot[bot] to identify PRs opened by the bot, then cross-check the PR audit trail for merged:true events without preceding review:approved events from human accounts. Alternatively, use GitHub's GraphQL API to query merged PullRequest objects where author is a bot and reviews contains no human approvals. Cross-check all dependency packages introduced via bot PRs against OSV.dev (https://osv.dev) and the npm advisory database for known malicious entries. Behavioral indicators in CI/CD: unexpected outbound network connections during dependency installation steps, execution of scripts not present in prior build runs, new environment variable reads or file writes outside expected build directories. In SIEM, alert on package manager processes (npm, pip, yarn) spawning child processes that initiate external connections during pipeline execution. Flag any dependency with a publication date within 7 days of the bot's update proposal, this is a high-confidence typosquatting indicator.

Framework Mappings

MITRE-ATTACK

- **T1059** — Command and Scripting Interpreter
- **T1072** — Software Deployment Tools
- **T1554** — Compromise Host Software Binary
- **T1195.001** — Compromise Software Dependencies and Development Tools

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **AT-2** — Literacy Training and Awareness
- **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
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **15.1** — Establish and Maintain an Inventory of Service Providers

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

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

SOC2-TSC

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059	Command and Scripting Interpreter	Execution
T1072	Software Deployment Tools	Execution
T1554	Compromise Host Software Binary	Persistence
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Sources

Source	URL	Tier
Renovate & Dependabot: The New Malware Delivery System	https://blog.gitguardian.com/renovate-dependabot-the-new-malware-de...	T3
Renovate integration with Dependabot security alerts for transitive ...	https://github.com/renovatebot/renovate/discussions/41825	T3

Source	URL	Tier
Renovate vs. Dependabot: Which Bot Will Rule Your Monorepo?	https://dev.to/alex_aslam/renovate-vs-dependabot-which-bot-will-rul...	T3
Security and Permissions - Renovate Docs	https://docs.renovatebot.com/security-and-permissions/	T3
A comprehensive study of Dependabot's impact on vulnerability ...	https://link.springer.com/article/10.1007/s10664-025-10638-w	T3

Generated 2026-04-18 18:46 UTC by TJS Security Command Center