INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-23 19:01 UTC

Laravel-Lang Supply Chain Compromise: 700+ Package Versions Weaponized to Drain Cloud Credentials, CI/CD Tokens, and Crypto Wallets

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0356
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	laravel-lang/lang, laravel-lang/http-statuses, laravel-lang/attributes, laravel-lang/actions (700+ malicious version tags); downstream exposure: Laravel, Symfony, PHPUnit applications using Composer autoloading
Published	2026-05-23T05:51:13
Discovery Source	Rss

Executive Summary

Between May 22-23, 2026, attackers compromised the Laravel-Lang GitHub organization and injected malicious code into 700+ version tags across four widely used PHP packages. Any PHP application that pulled these packages via Composer will auto-execute the payload on startup, exposing cloud credentials (AWS, Azure, GCP), CI/CD pipeline tokens, cryptocurrency wallets, browser-stored passwords, and SSH keys to an attacker-controlled server. The breach affects Laravel, Symfony, and PHPUnit ecosystems, meaning the scope of compromise extends across a significant portion of the PHP software supply chain.

Technical Analysis

Affected packages: laravel-lang/lang, laravel-lang/http-statuses, laravel-lang/attributes, laravel-lang/actions, 700+ malicious version tags published May 22-23, 2026. No CVE assigned as of May 23, 2026. CVSS base 9.5 is an editorial severity estimate pending NVD assignment. CWE mappings: CWE-494 (Download of Code Without Integrity Check), CWE-506 (Embedded Malicious Code), CWE-522 (Insufficiently Protected Credentials), CWE-312 (Cleartext Storage of Sensitive Information). Attack vector: Supply chain compromise of the Laravel-Lang GitHub organization (T1195.001). The embedded payload executes automatically via Composer autoloading on PHP application startup, no user interaction required. Payload behavior: enumerates and exfiltrates AWS IAM, GCP, and Azure credentials; CI/CD tokens (GitHub Actions, GitLab Runners, CircleCI, Jenkins, ArgoCD, TravisCI); Kubernetes/Helm configs; cryptocurrency wallets (Electrum, Exodus, Atomic,

MetaMask); browser credential stores (Chrome, Edge, Firefox, Brave, Opera); password manager vaults (1Password, Bitwarden, LastPass, KeePass); SSH/FTP client configs (PuTTY, WinSCP, FileZilla); messaging client tokens (Discord, Slack, Telegram, Outlook); VPN configs (OpenVPN, WireGuard, NordVPN, ExpressVPN); Docker credentials. Exfiltration is reported to use AES-256 encryption to a single C2 domain (details pending confirmation through malware analysis). Payload is reported to self-delete after exfiltration (T1070.004), though self-deletion behavior should be confirmed through analysis of affected systems or threat intelligence reports. MITRE techniques include T1195.001 (Supply Chain Compromise: Compromise Software Dependencies), T1528 (Steal Application Access Token), T1555/T1555.003 (Credentials from Password Stores / Web Browsers), T1552.001 (Credentials in Files), T1041 (Exfiltration Over C2 Channel), T1027 (Obfuscated Files or Information), T1059.004/T1059.005 (Command and Scripting Interpreter). No patch released from affected maintainers as of the discovery timestamp, confirmed clean versions must be identified by reviewing upstream GitHub tag history and Composer lock files.

Action Checklist

1. Step 1: Containment. Immediately audit composer.lock files across all PHP projects for any version of laravel-lang/lang, laravel-lang/http-statuses, laravel-lang/attributes, or laravel-lang/actions. Remove or pin these packages to confirmed-clean commits predating May 22, 2026. Take affected build pipelines offline until packages are validated. Block outbound connections to the known C2 domain at the perimeter firewall if the domain is identified through IOC feeds (see detection_guidance). Apply CIS Controls v8 2.2 (Address Unauthorized Software) and NIST SP 800-53 CM-3 (Change Control) and CM-5 (Access Restrictions) to prevent further Composer installs until packages are validated.
2. Step 2: Detection. Search CI/CD pipeline logs (GitHub Actions, GitLab Runner, CircleCI, Jenkins) for unexpected outbound HTTPS connections to unfamiliar domains during composer install or application startup, particularly between May 22-23, 2026. Query SIEM for process-level events showing PHP spawning child processes or making external network calls on startup. Check endpoint/server file system for evidence of self-deleted files (T1070.004) in temp directories. Review AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for credential use anomalies (new IPs, unusual API calls, new IAM key usage) originating from build or deploy systems. Cross-reference NIST AU-6 (Audit Record Review) and AU-12 (Audit Record Generation). Look for indicators matching T1555.003 (browser credential access) and T1552.001 (file-based credential reads) on developer workstations that ran affected builds.
3. Step 3: Eradication. Do not simply remove the package entries; assume all credentials stored on or accessible from affected systems have been compromised. Rotate all cloud IAM keys (AWS, GCP, Azure) used on systems that ran the compromised packages. Revoke and reissue all CI/CD tokens and pipeline secrets. Rotate SSH keys for servers accessible from affected build environments. Revoke and reissue API keys for all services (Docker Hub, container registries, Kubernetes service accounts). Apply NIST IA-5 (Authenticator Management) across all affected credential classes. Remove compromised package versions from all Composer caches and rebuild from confirmed-clean dependency trees. Apply CIS Controls v8 6.2 (Account Authorization and Deprovisioning) to any accounts whose credentials may have been resident on affected systems.
4. Step 4: Recovery. After credential rotation, validate that no new unauthorized access sessions are active in cloud consoles, CI/CD platforms, and Kubernetes clusters. Re-enable build pipelines only after composer.lock is pinned to verified-clean package hashes. Enable integrity verification (NIST SI-7: System Monitoring and File Integrity Verification) for package downloads going forward. Monitor for continued C2 beaconing or lateral movement indicators for at least 30 days. Confirm no persistence mechanisms were

installed during the payload execution window by running system configuration analysis checks on affected servers.

5. Step 5: Post-Incident. This attack exploited the absence of supply chain integrity controls (CWE-494). Implement Composer lock file hash verification (composer lock --check in CI/CD) and package signature validation aligned with NIST SA-12 and SA-15. Require code review of all composer.lock changes (CIS Controls v8 4.6). Enforce least privilege (NIST AC-6) on CI/CD pipeline credentials; pipelines should not have access to production cloud credentials they do not need at build time. Adopt NIST AC-2 (Account Management) and AC-6 controls to isolate CI/CD service accounts. Establish a third-party package monitoring process aligned with NIST SA-15 to detect future supply chain anomalies earlier.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and initiate formal breach notification assessment immediately if AWS CloudTrail, GCP Audit Logs, or Azure Activity Logs show any API call, authentication event, or resource access using credentials that were resident on a system that executed a laravel-lang package version tagged between May 22–23, 2026, as this constitutes confirmed credential exfiltration and likely triggers breach notification obligations under GDPR Article 33, CCPA, and applicable state data breach statutes.
Recovery Notes	After credential rotation and pipeline re-enablement, maintain elevated monitoring on all cloud IAM entities, CI/CD service accounts, and Kubernetes cluster roles that were accessible from affected build environments for a minimum of 30 days, as attackers with exfiltrated AWS, GCP, or Azure credentials may have established out-of-band persistence (new IAM users, rogue OAuth apps, Lambda backdoors) that survives the initial credential rotation. Verify recovery completeness by running `composer validate` and `composer audit` on every restored project and confirming zero laravel-lang entries in the May 22–23 tag range remain in any composer.lock across the environment. Any detection of C2 beacon traffic to the identified exfiltration domain after containment should be treated as evidence of a persisted implant and trigger a full reimaging cycle for the beaconing host.

Forensic Artifacts	composer.lock files from all affected PHP projects: record exact package name, version tag, dist URL, and dist.shasum for laravel-lang/lang, laravel-lang/http-statuses, laravel-lang/attributes, and laravel-lang/actions — these are the ground-truth artifacts proving which malicious version was installed and when | Composer cache tarballs at ~/.composer/cache/files/laravel-lang/ (Linux/macOS) or %APPDATA%/Composer/cache/files/laravel-lang/ (Windows): the cached malicious package archives may survive application cleanup and contain the full payload code for static malware analysis and IOC extraction | CI/CD pipeline raw job logs for the May 22–23, 2026 window from GitHub Actions, GitLab Runner, CircleCI, or Jenkins: these capture the exact moment composer install executed the malicious autoloader, including any stdout/stderr from the payload's outbound HTTP call to the C2 exfiltration endpoint | AWS CloudTrail event history filtered on AccessKeyId values for all IAM keys resident on affected build systems: specifically GetSecretValue, AssumeRole, CreateUser, AttachUserPolicy, and PutBucketPolicy events sourced from build server IPs between May 22 and present, which would confirm whether stolen credentials were weaponized for lateral movement or persistence in the AWS environment | PHP application /tmp and /var/tmp directory forensic recovery: the payload is documented as self-deleting (MITRE T1070.004), but Linux ext4 filesystem inode recovery via `debugfs -R 'lsdel' /dev/sdX` or the `extundelete` utility may recover the dropped payload binary or script, providing the full malware sample for hash-based IOC generation and submission to threat intelligence platforms

Per-Action IR Details

Step 1: Containment — Immediately audit composer.lock files across all PHP projects for any version of laravel-lang/lang, laravel-lang/http-statuses, laravel-lang/attributes, or laravel-lang/actions. Remove or pin these packages to confirmed-clean commits predating May 22, 2026. Take affected build pipelines offline until packages are validated. Block outbound connections to the known C2 domain at the perimeter firewall if the domain is identified through IOC feeds (see detection_guidance). Apply CIS 2.3 (Address Unauthorized Software) and NIST CM-3 principles to freeze further Composer installs until packages are vetted.

NIST Phase: Containment

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

Controls: NIST CM-3 (Configuration Change Control), NIST CM-7 (Least Functionality), NIST SI-3 (Malicious Code Protection), CIS 2.3 (Address Unauthorized Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run `grep -r 'laravel-lang' /path/to/projects --include='composer.lock' -l` across all project directories to enumerate affected lockfiles. For firewall blocking without enterprise tooling, add an iptables DROP rule: `iptables -A OUTPUT -d -j DROP` on each affected build server. On GitHub Actions/GitLab Runner hosts, set `COMPOSER_NO_INTERACTION=1` and revoke the runner's network egress via host-based firewall until the lockfile is pinned. Use `composer show laravel-lang/lang --all` to confirm installed version hash against the pre-May 22 commit SHAs from the official laravel-lang GitHub repository.

Evidence: Before removing any packages, snapshot the current state: (1) Copy all composer.lock and composer.json files verbatim — these record the exact malicious version tag pulled. (2) Capture the Composer cache directory (`~/.composer/cache/` or `%APPDATA%/Composer/cache/`) — the cached tarballs may contain the malicious payload for static analysis. (3) On CI/CD runners, dump active environment variables (`printenv > /tmp/env_snapshot_$(date +%s).txt`) before pipeline teardown — the payload targets env vars containing AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, GOOGLE_APPLICATION_CREDENTIALS, AZURE_CLIENT_SECRET, and CI token variables (GITHUB_TOKEN, GITLAB_TOKEN, CIRCLE_TOKEN). (4) Capture outbound netstat state: `ss -tnp | grep php` or `netstat -anp | grep php` to document any live C2 connections before blocking.

Step 2: Detection — Search CI/CD pipeline logs (GitHub Actions, GitLab Runner, CircleCI, Jenkins) for unexpected outbound HTTPS connections to unfamiliar domains during composer install or application startup, particularly between May 22–23, 2026. Query SIEM for process-level events showing PHP spawning

child processes or making external network calls on startup. Check endpoint/server file system for evidence of self-deleted files (T1070.004) in temp directories. Review AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for credential use anomalies (new IPs, unusual API calls, new IAM key usage) originating from build or deploy systems. Cross-reference NIST AU-6 (Audit Record Review) and AU-12 (Audit Record Generation). Look for indicators matching T1555.003 (browser credential access) and T1552.001 (file-based credential reads) on developer workstations that ran affected builds.

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 AU-3 (Content of Audit Records), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Without SIEM: (1) On GitHub Actions, download workflow run logs via `gh run list --limit 50 --json databaseId,createdAt | jq '.[].databaseId'` then `gh run view --log` and grep for unexpected curl/wget/file_get_contents calls or unfamiliar domains. (2) Deploy Sysmon with SwiftOnSecurity config and filter on Event ID 3 (Network Connection) where Image ends in `php.exe` or `php` — any connection to a non-local, non-package-registry IP on port 443 during startup is suspicious. (3) For AWS CloudTrail without SIEM, use `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetSecretValue --start-time 2026-05-22T00:00:00Z --end-time 2026-05-24T00:00:00Z` to surface any secret reads from build system IPs. (4) Use osquery: `SELECT pid, name, cmdline, parent FROM processes WHERE name LIKE '%php%' AND cmdline LIKE '%curl%' OR cmdline LIKE '%file_get_contents%';`

Evidence: Capture before analysis: (1) PHP application startup logs and web server error logs (`/var/log/nginx/error.log`, `/var/log/apache2/error.log`, `/var/log/php*.log`) for the May 22–23, 2026 window — the payload executes on autoload, so any error or HTTP client call during `composer install` or framework bootstrap is a direct indicator. (2) CI/CD raw job logs for all pipeline runs between May 22 00:00 UTC and May 23 23:59 UTC — look for outbound HTTPS POSTs in the composer install step. (3) AWS CloudTrail `AssumeRole`, `GetCallerIdentity`, and `ListBuckets` events sourced from build server IPs or Lambda ARNs used by the pipeline. (4) On developer workstations: `~/.ssh/known_hosts`, `~/.aws/credentials`, `~/.config/gcloud/application_default_credentials.json`, and browser SQLite credential stores (`Login Data` on Chrome/Edge at `%LOCALAPPDATA%\Google\Chrome\User Data\Default\`) — these are the exact file paths the T1552.001 and T1555.003 TTPs target. (5) Sysmon Event ID 3 (Network Connection) and Event ID 1 (Process Create) showing PHP process tree with unexpected child processes.

Step 3: Eradication — Do not simply remove the package entries; treat all credentials stored on or accessible from affected systems as fully compromised. Rotate all cloud IAM keys (AWS, GCP, Azure) used on systems that ran the compromised packages. Revoke and reissue all CI/CD tokens and pipeline secrets. Rotate SSH keys for servers accessible from affected build environments. Revoke and reissue API keys for all services (Docker Hub, container registries, Kubernetes service accounts). Apply NIST IA-5 (Authenticator Management) and D3-CRO (Credential Rotation) across all affected credential classes. Remove compromised package versions from all Composer caches and rebuild from confirmed-clean dependency trees. Apply CIS 6.2 (Access Revoking Process) to any accounts whose credentials may have been resident on affected systems.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IA-5 (Authenticator Management), NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), NIST CM-3 (Configuration Change Control), CIS 6.2 (Establish an Access Revoking Process), CIS 5.1 (Establish and Maintain an Inventory of Accounts)

Compensating: Without an enterprise secrets management platform: (1) AWS key rotation: `aws iam create-access-key --user-name ` then immediately `aws iam delete-access-key --access-key-id ` — do not delete before creating the new key or pipeline breaks. (2) GitHub Actions: navigate to Settings → Secrets and variables → Actions, delete each secret and re-add from a clean credential source. (3) SSH key rotation: generate new keypairs with `ssh-keygen -t ed25519 -C 'rotated-$(date +%Y%m%d)'`, push public keys to `~/.ssh/authorized_keys` on all target servers, remove old public key entries, and verify with `ssh -i user@host`. (4) Clear Composer cache on all

affected CI runners: `composer clear-cache` or `rm -rf ~/.composer/cache/` — the cached malicious tarballs will otherwise persist. (5) Audit Kubernetes service account tokens: `kubectl get secrets --all-namespaces | grep 'service-account-token'` and rotate any exposed.

Evidence: Before rotating credentials, preserve forensic copies: (1) Export the full IAM access key usage history from AWS CloudTrail — `aws cloudtrail lookup-events --lookup-attributes AttributeKey=AccessKeyId,AttributeValue=` — this establishes attacker dwell time and scope of API calls made with stolen credentials. (2) Snapshot the Kubernetes audit log (`/var/log/kube-apiserver-audit.log`) for service account activity between May 22–23, 2026. (3) Export CI/CD pipeline secret access audit logs from GitHub (Settings → Security log, filter `secret_scanning`) and GitLab (Admin Area → Audit Events) before revoking tokens, as revocation may purge log context in some platforms. (4) Preserve a disk image or at minimum a file listing of `/tmp`, `/var/tmp`, and the PHP application root — the payload is documented as self-deleting (T1070.004) but forensic recovery of deleted inodes via `debugfs` or `extundelete` may recover the dropped file for malware analysis.

Step 4: Recovery — After credential rotation, validate that no new unauthorized access sessions are active in cloud consoles, CI/CD platforms, and Kubernetes clusters. Re-enable build pipelines only after composer.lock is pinned to verified-clean package hashes. Enable integrity verification (NIST SI-7, D3-FMBV — File Magic Byte Verification) for package downloads going forward. Monitor for continued C2 beaconing or lateral movement indicators for at least 30 days. Confirm no persistence mechanisms were installed during the payload execution window by running D3-SICA (System Init Config Analysis) checks on affected servers.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CP-10 (System Recovery and Reconstitution), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Without enterprise integrity monitoring: (1) Pin all four affected packages in composer.json to the last confirmed-clean Git commit SHA predating May 22, 2026 using `composer require laravel-lang/lang:#` syntax and commit the resulting composer.lock to version control with a mandatory reviewer. (2) Add a pre-commit hook that runs `composer validate --strict` and `php -r "require 'vendor/autoload.php';"` and fails if any laravel-lang package resolves to a tag in the May 22–23 window. (3) For persistence checks without EDR, run `systemctl list-units --type=service | grep -v default` and `crontab -l -u www-data` and `ls -la /etc/cron.d/` on each affected server. (4) For 30-day beacon monitoring, deploy Zeek or run `tcpdump -nn -i eth0 'tcp port 443' -w /tmp/cap_$(date +%Y%m%d).pcap` daily on build server egress interfaces and review for repeated connections to the C2 domain.

Evidence: Before re-enabling pipelines, document the verified-clean state: (1) Record the exact commit SHA of each laravel-lang package pinned in the restored composer.lock and store this in your change record as the integrity baseline. (2) Run `php -r "echo md5_file('vendor/laravel-lang/lang/src/Lang.php');"` (adjust path per package) on a known-clean installation and record the hash as a reference value for future integrity checks. (3) Capture a post-rotation AWS CloudTrail snapshot for 24 hours post-recovery to confirm no residual use of the old compromised access keys — any hit on the old key ID after rotation confirms the attacker retained and is using the credential. (4) Query Kubernetes audit logs for any new `ClusterRoleBinding` or `RoleBinding` objects created between May 22 and recovery date — these would indicate attacker persistence via privilege escalation using stolen service account tokens.

Step 5: Post-Incident — This attack exploited the absence of supply chain integrity controls (CWE-494). Implement Composer hash-locking and artifact signing (NIST SA-12, SA-15) for all third-party PHP dependencies. Require code review of composer.lock changes in pull requests (CIS 4.6 — Securely Manage Enterprise Assets and Software). Enforce least privilege (NIST AC-6) on CI/CD pipeline credentials — pipelines should not have access to production cloud credentials they do not need at build time. Adopt D3-UAP (User Account Permissions) controls to isolate CI/CD service accounts. Establish a third-party package monitoring process aligned with CIS 7.1 (Vulnerability Management Process) to detect future supply chain anomalies earlier.

NIST Phase: Post Incident

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

Controls: NIST SA-12 (Supply Chain Protection), NIST SA-15 (Development Process, Standards, and Tools), NIST AC-6 (Least Privilege), NIST AC-3 (Access Enforcement), NIST RA-3 (Risk Assessment), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Without enterprise supply chain tooling: (1) Enable Composer's built-in hash verification by ensuring every package entry in composer.lock has a populated `dist.shasum` field — `composer install --no-dev` will verify these hashes on every install with no additional tooling. (2) Subscribe to the GitHub repository RSS feed for each laravel-lang package (`https://github.com/laravel-lang/lang/releases.atom`) and route alerts to your security inbox for new tag creation events. (3) Write a GitHub Actions workflow step that runs `composer audit` (built into Composer 2.4+) and fails the pipeline if any advisory is returned for laravel-lang packages. (4) For CI/CD least privilege: scope GitHub Actions OIDC tokens to the minimum required permissions by setting `permissions: { contents: read }` in workflow YAML, eliminating long-lived static credentials entirely. (5) Use YARA rule targeting `file_get_contents` or `curl_exec` calls within Composer vendor directories as a lightweight supply chain malware scan on each build.

Evidence: Lessons-learned documentation artifacts to capture: (1) The complete list of composer.lock files across all projects that contained any of the four affected package names, with version tags and pull timestamps — this defines the full blast radius for regulatory notification purposes. (2) A timeline mapping the first malicious composer install per project to the first detected anomalous cloud API call per credential set — this establishes attacker dwell time per affected system for the post-incident report. (3) AWS IAM Access Advisor reports for all rotated IAM users/roles showing last-accessed service data — this documents whether stolen credentials were actually used for AWS API calls, which is material for breach scope determination under SOC 2 and PCI DSS requirements.

Detection Guidance

Primary detection surface is CI/CD pipeline and build server logs from the May 22-23, 2026 window. Query for: (1) Outbound HTTPS connections from PHP processes or Composer execution contexts to non-standard domains, particularly during composer install or application bootstrap. (2) Process tree anomalies where php or php-fpm spawned shell processes (sh, bash, cmd), relevant to T1059.004/T1059.005. (3) File system events showing creation and deletion of temporary files in /tmp, %TEMP%, or application working directories during startup, self-deletion indicator for T1070.004. (4) Credential file reads: access to ~/.aws/credentials, ~/.config/gcloud/, ~/.azure/, ~/.kube/config, ~/.ssh/, browser profile directories (Chrome Default/Login Data, Firefox profiles/logins.json), or password manager vault files by PHP processes. (5) Abnormal network egress from build servers or application servers to single unfamiliar domains, consistent with encrypted exfiltration over single C2 (T1041). (6) New or anomalous IAM key usage in AWS CloudTrail (event: ConsoleLogin, CreateAccessKey, AssumeRole from new source IPs), GCP Cloud Audit Logs, and Azure Sign-In Logs correlated to times when affected packages were installed. IOC LIMITATION: As of May 23, 2026, the specific C2 domain and IP address have not been disclosed in public threat intelligence feeds. Organizations must monitor Laravel-Lang GitHub repository, CISA advisories, and VirusTotal for IOC updates before implementing perimeter blocks. Until IOCs are confirmed, rely on behavioral detection (outbound HTTPS anomalies, credential access patterns) as primary detection surface. NIST AU-6 and AU-12 controls support the log review and correlation process.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`Not publicly confirmed in available sources as of May 23, 2026`	Single C2 domain used for AES-256 encrypted credential exfiltration — monitor threat intelligence feeds and Laravel-Lang GitHub security advisories for disclosure	LOW
URL	`https://github.com/Laravel-Lang/http-statuses`	One of four compromised package repositories — use to identify malicious version tags published May 22–23, 2026	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1528** — Steal Application Access Token
- **T1082** — System Information Discovery
- **T1555.003** — Credentials from Web Browsers
- **T1078.004** — Cloud Accounts
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1555** — Credentials from Password Stores
- **T1041** — Exfiltration Over C2 Channel
- **T1552.001** — Credentials In Files
- **T1070.004** — File Deletion
- **T1027** — Obfuscated Files or Information
- **T1083** — File and Directory Discovery
- **T1552.004** — Private Keys
- **T1588.006** — Vulnerabilities
- **T1539** — Steal Web Session Cookie
- **T1059.005** — Visual Basic
- **T1059.004** — Unix Shell

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **SI-3** — Malicious Code Protection
- **CM-7** — Least Functionality
- **IA-5** — Authenticator Management
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **5.2** — Use Unique Passwords
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication
- **164.312(e)(1)** — Transmission Security

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

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
- **A.8.24** — Use of cryptography
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1528	Steal Application Access Token	Credential-Access
T1082	System Information Discovery	Discovery
T1555.003	Credentials from Web Browsers	Credential-Access
T1078.004	Cloud Accounts	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Technique ID	Technique Name	Tactic
T1555	Credentials from Password Stores	Credential-Access
T1041	Exfiltration Over C2 Channel	Exfiltration
T1552.001	Credentials In Files	Credential-Access
T1070.004	File Deletion	Defense-Evasion
T1027	Obfuscated Files or Information	Defense-Evasion
T1083	File and Directory Discovery	Discovery
T1552.004	Private Keys	Credential-Access
T1588.006	Vulnerabilities	Resource-Development
T1539	Steal Web Session Cookie	Credential-Access
T1059.005	Visual Basic	Execution
T1059.004	Unix Shell	Execution

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/laravel-lang-php-packages-comprom...	T3
Translation of HTTP statuses - Laravel Lang - GitHub	https://github.com/Laravel-Lang/http-statuses	T3
The Ease of Deployment Tier List for Laravel Developers - YouTube	https://www.youtube.com/watch?v=0VJaUcjVjWo	T3
Your Laravel Ecosystem on Top of the World - Fly.io	https://fly.io/laravel/	T3
HTTP Statuses - Laravel Lang	https://laravel-lang.com/packages-http-statuses.html	T3

Generated 2026-05-23 19:01 UTC by TJS Security Command Center