WHITE PAPER

CYTEX™
AI-Powered Resilience Command Center

Laravel Lang Compromised

With RCE Backdoor Across 700+ Package Versions

hello@cytex.io | https://cytex.io

Laravel Lang Compromised with RCE Backdoor Across 700+ Package Versions Through GitHub Cross-Fork Tag Abuse

Technical analysis of the May 22 and 23, 2026 supply chain compromise of the Laravel Lang ecosystem, exploiting a structural ambiguity in how Composer resolves GitHub tag references.

What Happened

The community-maintained Laravel Lang project was compromised with a remote code execution backdoor spanning hundreds of package versions across multiple related packages. The affected packages include laravel-lang/lang, laravel-lang/http-statuses, laravel-lang/attributes, and laravel-lang/actions. Approximately 700 historical versions across these packages are affected.

The attack is structurally distinct from a typical compromised-package incident. No malicious code was ever committed to the official Laravel Lang repositories. The attacker did not gain write access to the official codebase. Instead, the attacker exploited a permissive behavior in how GitHub allows version tags to point to commits, and how Composer resolves those tag references to install package code.

Tags were published in rapid succession on May 22 and May 23, 2026, with many versions appearing only seconds apart. Applications that installed compromised versions executed the backdoor automatically the next time Composer's autoloader ran, with no user interaction required.

How the Tag Hijack Works

The attack exploits a property of GitHub's tag model: a version tag in a repository can point to a commit that lives in a fork of that repository, not only to commits in the upstream's own history. From Composer's perspective, when a tag is published in laravel-lang/lang, the package manager resolves the tag, pulls the commit it references, and installs that commit's contents as the package source.

The attacker created a malicious fork of the Laravel Lang repository, placed the backdoor code in their fork, and published tags in the official repository that pointed to commits in the malicious fork. Composer followed the tag references, fetched the attacker-controlled commits, and installed the backdoor into downstream applications.

This bypasses every defensive control that assumes "if the official repository's commit history is clean, the package is clean":

- Repository audits of the official Laravel Lang codebase show no malicious commits, because no malicious code was committed there

- Branch protection on the official repository is irrelevant, because the attacker never wrote to the official repository
- Pull request review processes are irrelevant, because no pull request was opened against the official repository
- Maintainer credential security is irrelevant for the commit history, because the attacker did not need maintainer credentials to commit code

The tagging operation itself does require some level of access to the official repository. Whether the attacker compromised a maintainer account, exploited a permission misconfiguration, or used another mechanism to publish tags has not been fully publicly characterized in the available reporting. The structural defect being exploited is the cross-fork tag-to-commit resolution behavior, which is documented and permitted by GitHub but rarely treated as a security-relevant property by package consumers.

Payload Execution

The malicious code executes automatically when an affected application loads, with no explicit invocation required:

Composer autoload trigger. Each affected package version includes a malicious `src/helpers.php` file. The package's `composer.json` declares this file under the autoload.files directive, which instructs Composer to require the file every time the autoloader is initialized. Every PHP request, every CLI command, every queue worker startup, and every artisan invocation triggers the malicious file.

Permission inheritance. The backdoor runs with the full permissions of the host web application. On a typical Laravel deployment, this includes filesystem read and write access to the application directory, database access through configured connections, network egress, environment variable visibility, and access to any credentials the application is configured to use.

Audit evasion. Because no malicious code lives in the official repository's main or master branch, code audits of the upstream project do not surface the backdoor. The malicious file only appears in the installed vendor/ directory of downstream applications, and only after the affected version has been resolved and installed.

Initial Dropper Behavior

The `src/helpers.php` file presents itself as a routine Laravel localization helper. Inspection by name or surface signature does not flag the file as suspicious. The dropper's actual behavior:

- **Host fingerprinting.** The dropper collects hardware and environment metrics from the host system, including the application directory path, system architecture, and filesystem inode information.

- **Per-host marker generation.** It generates an MD5 hash combining the collected fingerprint values. This hash serves as a unique marker for each installation.
- **Execute-once logic.** The marker is checked against a local record before the secondary payload is fetched. If the marker indicates the dropper has already executed on this host, subsequent invocations exit silently. This prevents repeated network activity that would draw attention and reduces the chance of behavioral detection across multiple application restarts.

Secondary Payload Retrieval

If the dropper has not previously executed on the host, it proceeds to fetch and run the credential-stealing payload:

- **SSL verification disabled.** The dropper disables SSL/TLS certificate verification before initiating its outbound connection. This prevents certificate-based blocking and allows the command-and-control connection to succeed even where the attacker controls only an obfuscated endpoint with a non-standard certificate.
- **Obfuscated command-and-control fetch.** The secondary script is retrieved from a command-and-control server reached through an obfuscated URL or address resolution path. The specific obfuscation mechanism varies across the affected versions.
- **Silent execution.** The fetched secondary payload is launched via OS-specific methods that avoid creating visible processes, console windows, or log entries on the host.

Credential Stealer Module Set

The secondary payload deploys **15 distinct credential collector modules**, targeting categories of secrets commonly present on developer and application server systems:

Cloud access credentials:

- AWS access keys and session tokens from standard credential file locations
- Google Cloud Platform service account keys and gcloud configuration
- Azure credentials and CLI configuration
- DigitalOcean API tokens

Infrastructure configuration:

- Kubernetes kubeconfig profiles, including embedded tokens and certificates
- Docker tokens, registry credentials, and configuration files
- HashiCorp Vault secrets and authentication artifacts

Developer assets:

- SSH private keys from `~/.ssh/` and equivalent locations
- Git credentials, including stored authentication tokens for GitHub, GitLab, and Bitbucket
- Shell history files containing command-line credential exposure

Application and personal assets:

- Saved browser passwords from local browser storage
- Cryptocurrency wallet files and associated keys
- Password manager databases

Application configuration:

- Database connection strings from .env files and equivalent configuration
- Cloud metadata from local metadata endpoints reachable by the application
- Application environment variables loaded into the running PHP process

Exfiltration Behavior

After harvesting, the payload encrypts the collected data using **AES-256** before transmission. Encrypted exfiltration is sent to the attacker's infrastructure, after which the payload **deletes itself from the host** to remove forensic evidence of its execution. The initial src/helpers.php dropper remains in the application's vendor/ directory and continues to mark subsequent installations as already-processed, but the secondary payload that performed the credential collection leaves minimal residual artifacts on the host.

Indicators of Compromise

Affected packages:

- laravel-lang/lang
- laravel-lang/http-statuses
- laravel-lang/attributes
- laravel-lang/actions

Affected version count: Approximately 700 historical versions across the four packages, tagged on May 22 and May 23, 2026.

Composer-level indicators:

- Presence of any of the affected packages at versions tagged during the May 22-23, 2026 window in composer.lock
- A `src/helpers.php` file in the installed package that includes Laravel-style helper function names but executes outbound network or fingerprinting logic
- Composer-installed package contents that differ from the contents of the corresponding tag in the official Laravel Lang upstream repository

Host-level indicators:

- Outbound HTTPS connections from PHP-FPM, php-cli, or web server processes to destinations not associated with the application's normal traffic
- TLS connections with disabled certificate verification originating from the PHP process tree
- Reads against credential locations `including ~/.aws/credentials, ~/.config/gcloud/, ~/.azure/, ~/.kube/config, ~/.ssh/id_*`, and `.env` files by the PHP process
- Access to browser credential databases and cryptocurrency wallet files by processes spawned from the web application context

Repository-level indicators (for forensic analysis):

- Tag references in the official Laravel Lang repository pointing to commit SHAs that are not in the official repository's commit history
- Tag creation events on the official repository during the May 22-23, 2026 window with commit references resolving to non-upstream forks

Defensive Actions

Rotate all reachable credentials. Any application that installed an affected version should be treated as having exposed every secret the running process had access to. Rotate AWS, GCP, Azure, DigitalOcean credentials. Rotate database credentials, API keys, OAuth client secrets, and any credentials present in .env files. Rotate SSH keys for hosts that ran affected applications. Audit cloud provider access logs for unauthorized API calls during the exposure window.

Block affected package versions in dependency resolution. Add explicit exclusions for the affected version ranges in composer.json constraint definitions and in any private package proxy or repository policy enforcement. Re-resolve composer.lock and verify the installed versions.

Rebuild compromised hosts from known-good images. Self-deletion of the secondary payload removes some artifacts but does not provide assurance that no persistence was established. Hosts confirmed to have executed an affected package version should be rebuilt rather than cleaned in place. Production systems, CI runners, and developer workstations are all in scope.

Audit outbound network traffic for the exposure window. Review egress logs for unexpected destinations from PHP processes and any host that ran the affected applications. The exposure window opens at the package install time and continues until rotation and rebuild are complete.

Treat cross-fork tag resolution as a category-level risk. The structural defect this attack exploits is not specific to Laravel Lang or to PHP. Any package manager that resolves tags to commits through systems that permit cross-fork tag-to-commit mapping is exposed to the same attack pattern. Defensive programs should consider validating that installed package contents match the corresponding tag's contents in the upstream repository as part of the dependency verification pipeline.

Apply egress restrictions to application servers and developer environments. The payload's exfiltration channel relies on outbound HTTPS to attacker-controlled infrastructure. Network egress allowlists on application servers and developer workstations limit the destinations a compromised package can reach, and disabled SSL verification on the payload side does not help the attacker if the connection cannot be established in the first place.