

# Malicious Packagist Packages Disguised as Laravel Utilities

...

---

 [socket.dev/blog/malicious-packagist-packages-disguised-as-laravel-utilities](https://socket.dev/blog/malicious-packagist-packages-disguised-as-laravel-utilities)

[Back](#)

## Malicious Packagist Packages Disguised as Laravel Utilities Deploy Encrypted RAT

---

Malicious Packagist packages disguised as Laravel utilities install an encrypted PHP RAT via Composer dependencies, enabling remote

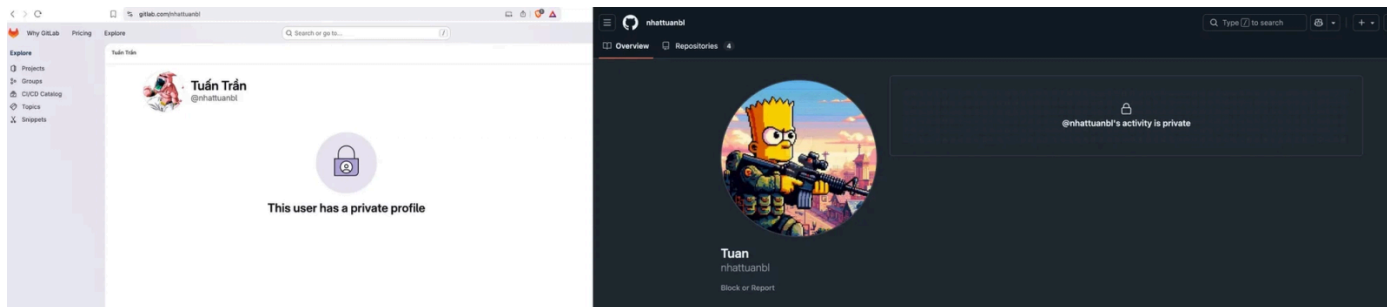
access and C2 callbacks.

---



Summarize with AI

Socket's Threat Research Team identified a remote access trojan (RAT) distributed across multiple Packagist (PHP) packages published by the threat actor [nhattuanbl](#) ([nhattuanbl@gmail.com](mailto:nhattuanbl@gmail.com)). Two packages, [nhattuanbl/lara-helper](#) and [nhattuanbl/simple-queue](#), ship an identical payload in [src/helper.php](#). A third package, [nhattuanbl/lara-swagger](#), carries no malicious code itself but lists [nhattuanbl/lara-helper](#) as a hard Composer dependency, meaning that installing it pulls in the RAT automatically.



Threat actor *nhattuanbl*'s GitLab profile (left) displays "This user has a private profile." Their GitHub profile (right) similarly shows "@nhattuanbl's activity is private."

Once loaded, the payload connects to a C2 server at *helper[.]leuleu[.]net:2096*, sends system reconnaissance data, and waits for commands - giving the operator full remote access to the host. We've submitted takedown requests to the Packagist team; the packages remain live at the time of publication.

## The Packages

|                                                                                    |                |     |         |
|------------------------------------------------------------------------------------|----------------|-----|---------|
| nhattuanbl member since: Dec 12, 2015                                              |                |     |         |
| nhattuanbl's packages                                                              |                |     |         |
| <b>nhattuanbl/lara-helper</b><br>Just some helper functions & commands for Laravel | CONTAINS RAT   |     | 37<br>0 |
| <b>nhattuanbl/lara-media</b><br>Laravel media manager / convert                    | CLEAN          | PHP | 5<br>0  |
| <b>nhattuanbl/lara-swagger</b><br>Swagger ui doc for laravel                       | DEPENDS ON RAT |     | 49<br>0 |
| <b>nhattuanbl/simple-queue</b><br>Simple Async Queue with Redis                    | CONTAINS RAT   |     | 29<br>0 |
| <b>nhattuanbl/snooze</b><br>Avoid laravel notification spamming                    | CLEAN          | PHP | 11<br>0 |
| <b>nhattuanbl/syslog</b><br>Laravel model & request logging                        | CLEAN          | PHP | 88<br>0 |

*nhattuanbl*'s Packagist profile, member since December 12, 2015. *lara-helper* and *simple-queue* contain the RAT payload. *lara-swagger* depends on *lara-helper*. The remaining three packages are clean.

The same threat actor has also authored three clean packages that follow a similar naming pattern, which may lend legitimacy to the malicious ones. All six packages were published within a six-month window between June and December 2024, despite the Packagist account dating back

to December 2015. The RAT-bearing packages , `lara-helper` and `simple-queue` appeared last in that sequence, after the clean packages had already accumulated a small install count under the same author name.

`lara-syslog` references a class from `lara-helper` in its source but does not declare it as a Composer dependency - the import would fail at runtime unless `lara-helper` was installed separately. Similarly, `lara-media` contains a dead import of `FileHelper` from `lara-helper` and a local path repository hint ("`../lara-helper`") in its `composer.json`, though it does not declare `lara-helper` as a dependency. These dangling references across multiple packages suggest the threat actor was developing with `lara-helper` as a shared local dependency, meaning the RAT was present in their development environment even while working on otherwise clean packages.

## The Payload

---

Both `lara-helper` and `simple-queue` contain `src/helper.php`. The files are byte-for-byte identical.

**SHA-256:** `a493ce9509c5180e997a04cab2006a48202afbb8edfa15149a4521067191ead7`

**Size:** 27,340 bytes, delivered as a single line after the opening `<?php` tag.

## Obfuscation

---

The payload uses three layered techniques to resist static analysis:

- **goto spaghetti:** Control flow is broken into hundreds of labeled jumps with randomized names (`tc0pE`, `IlaiV`, `rAJ4a`). There is no readable top-to-bottom execution path.
- **Hex/octal string encoding:** Every string literal - domain names, command names, file paths - is encoded using `\\xNN` or `\\NNN` escape sequences. Nothing meaningful appears in plaintext.
- **Randomized identifiers:** All variable and function names are randomly generated (`$riz07`, `BsYhQ()`, `TB0sT()`).

## Activation

---

How the payload loads depends on which package pulls it in.

In `lara-helper`, the package registers a Laravel service provider via Composer's auto-discovery mechanism (`extra.laravel.providers`). The provider's `register()` method runs `include_once __DIR__ . '/helper.php'` on every application boot.

In `simple-queue`, the `include_once` sits at file scope in `src/SimpleQueue.php`. It executes the moment PHP's autoloader loads the file - the class does not need to be instantiated. Any code path that causes the autoloader to resolve `nhattuanbl\\SimpleQueue` - a type-hint, an

`instanceof` check, even `class_exists()` is enough to trigger the payload.

## Self-Launch

---

When `helper.php` is first included, it checks whether it is running with CLI arguments. If not, it immediately spawns a detached background copy of itself and passes it the argument `helper`:

```
# Unix
php /path/to/helper.php helper > /dev/null 2>&1 &

# Windows
start /B php /path/to/helper.php helper > NUL 2>&1
```

The spawned process enters the main RAT loop. The parent include returns, and normal application execution continues. From the application's perspective, nothing unusual happened.

A lock file at `{sys_get_temp_dir}/wvIjjnDMRaomchPprDBzzVSpzh61RCar.lock` prevents multiple instances from running at once. It expires after 15 minutes.

## C2 Communication

---

The RAT connects to `helper[.]leuleu[.]net` on port `2096` over raw TCP using PHP's `stream_socket_client()`. It retries indefinitely on failure, sleeping 15 seconds between attempts.

All traffic is encrypted:

- **Algorithm:** AES-128-CTR
- **Key:** `esCAmxUoJkIjTV0n` (hardcoded, 16 bytes)
- **IV:** 16 random bytes per message
- **Wire format:** `[4-byte length][16-byte IV][encrypted JSON]`

The key is hardcoded in the payload, so traffic can be decrypted by anyone who extracts it.

## Reconnaissance

---

On first connection, the RAT collects and transmits a system profile:

- **os\_type** - `PHP_OS`, resolves to "Windows", "Mac", or "Linux"
- **version** - OS version (`ver`, `sw_vers`, `/etc/os-release`)
- **name** - OS name
- **pc\_name** - Hostname via `php_uname('n')`
- **uid** - Machine unique ID (Windows registry GUID, Mac Hardware UUID, Linux `/etc/machine-id`)
- **permission** - Current user, UID, GID
- **dir** - Directory path of `helper.php`

- **vid - 1.2** (RAT version)

## Command Set

---

The C2 sends JSON objects with a command field (**s**). Supported commands:

| Command           | Behavior                                                                 |
|-------------------|--------------------------------------------------------------------------|
| <b>ping</b>       | Heartbeat, sent automatically every 60 seconds                           |
| <b>info</b>       | Return full system reconnaissance to C2                                  |
| <b>cmd</b>        | Run a shell command, return stdout                                       |
| <b>powershell</b> | Run a PowerShell command, return stdout                                  |
| <b>run</b>        | Run a shell command in the background, no output returned                |
| <b>screenshot</b> | Capture the screen using <b>imagegrabscreen()</b> , return as base64 PNG |
| <b>download</b>   | Read a file from disk, return contents as base64                         |
| <b>upload</b>     | Write a file to disk from base64 data, set permissions to <b>0777</b>    |
| <b>stop</b>       | Close the socket and exit                                                |

For shell execution, the RAT probes **disable\_functions** and picks the first available method from: **popen**, **proc\_open**, **exec**, **shell\_exec**, **system**, **passthru**. This makes it resilient to common PHP hardening configurations.

## The Dependency Chain Vector

---

**lara-swagger** is worth calling out separately because it contains no malicious code of its own. Its [composer.json](#) simply declares:

**lara-swagger**'s **composer.json** explicitly requires **nhattuanbl/lara-helper** at **dev-master**. Installing this package silently pulls in the RAT.

Running **composer require nhattuanbl/lara-swagger** silently installs **lara-helper** alongside it. The **dev-master** constraint always pulls the latest commit from the default branch, meaning the operator could update the payload at any time without touching **lara-swagger** itself.

This pattern, using a clean-looking utility package to pull in a malicious one, is an effective way to slip past reviews focused only on the directly installed package.

## Impact

---

Any Laravel application that installed `lara-helper` or `simple-queue` is running a persistent RAT. The threat actor has full remote shell access, can read and write arbitrary files, and receives an ongoing system profile for each connected host. The payload works on Windows, macOS, and Linux.

Because activation happens at application boot (via service provider) or class autoload (via `simple-queue`), the RAT runs in the same process as the web application with the same filesystem permissions and environment variables, including database credentials, API keys, and `.env` contents.

The C2 host at `helper[.]leuleu[.]net:2096` is not responding at the time of writing. This does not mean affected hosts are safe. The RAT is still present on disk and retries the connection every 15 seconds indefinitely. The operator can redirect it to a new host without touching the payload.

## Outlook and Recommendations

---

The `lara-helper` and `simple-queue` packages ship a fully functional RAT inside a file named `helper.php`, a name that blends naturally into a Laravel helper library. The obfuscation is heavy enough that a quick manual review of the package would likely miss it. `lara-swagger` extends the reach of the payload without containing any malicious code itself, making it a clean-looking vector that depends on a dirty one.

Any package that loads a PHP file unconditionally at boot or autoload time is a viable delivery mechanism. Transitive dependencies deserve the same scrutiny as direct installs, and `dev-master` constraints should be treated as particularly high-risk in production since they bypass version pinning entirely.

### If you installed any of these packages:

- Treat the host as compromised. Assume the operator has had full shell access since installation.
- Rotate all secrets accessible from the application environment (database passwords, API keys, `.env` values).
- Remove the packages and the `helper.php` payload. Check for uploaded files with `chmod 0777`.
- Delete the lock file at `{sys_get_temp_dir}/wvIjjnDMRaomchPprDBzzVSpzh61RCar.lock`.
- Audit outbound traffic to `helper[.]leuleu[.]net:2096`.

### For security teams:

- Flag packages that include non-composer PHP files loaded unconditionally at boot.

- Review transitive dependencies - the direct package you install may be clean while a required dependency is not.
- **dev-master** constraints on Packagist packages bypass version pinning and should be treated as high-risk in production environments.

*Socket's scanner flags **nhattuanbl/lara-helper** as known malware, describing the payload as a fully featured backdoor with an encrypted C2 channel, remote shell execution, file upload/download, and screen capture capabilities.*

Socket detects threats like this across the full dependency graph, including transitive installs, before they reach production. The [Socket GitHub App](#) scans pull request dependency changes and flags suspicious package introductions, including new files like **helper.php** that are loaded at boot before merge. The [Socket CLI](#) enforces allow and deny rules in CI pipelines, stopping risky dependency changes early. [Socket Firewall](#) blocks known malicious packages, including transitive dependencies, before they are fetched. The [Socket browser extension](#) surfaces risk signals during package evaluation on Packagist. [Socket MCP](#) prevents AI-assisted coding workflows from introducing suspicious or hallucinated dependencies into your projects.

## Indicators of Compromise (IOCs)

---

### Malicious Packages

---

- [nhattuanbl/lara-helper](#)
- [nhattuanbl/simple-queue](#)
- [nhattuanbl/lara-swagger](#)

### Threat Actor Profile

---

- **Registered name** - **nhattuanbl**
- **Registered email address** - **nhattuanbl@gmail[.]com**
- **Gitlab Profile** - **https://gitlab[.]com/nhattuanbl**
- **Github Profile** - **https://github[.]com/nhattuanbl**

### Network Indicators

---

**helper[.]leuleu[.]net:2096**

### Files Hashes

---

**src/helper.php -  
a493ce9509c5180e997a04cab2006a48202afbb8edfa15149a4521067191ead7**

## MITRE ATT&CK

---

- T1195.001 — Compromise Software Dependencies and Development Tools
- T1059.004 — Unix Shell
- T1059.001 — PowerShell
- T1027 — Obfuscated Files or Information
- T1571 — Non-Standard Port
- T1041 — Exfiltration Over C2 Channel
- T1105 — Ingress Tool Transfer
- T1113 — Screen Capture

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email