Laravel

Application Security Checklist

Laravel Application Security Checklist

Document Version 1

Latest version always available on:

laravelsecuritychecklist.com

Twitter: @darren_chowles

LinkedIn: Darren Chowles

Cover Design: Michael Macneil – ID Studio

Contributor: Stephen Rees-Carter – Securing Laravel

Table of Contents

Introduction

Welcome to the Laravel Application Security Checklist. My goal with this checklist is to provide a go-to for any Laravel application developer who needs to confirm they've taken some of the most important precautions to secure their web application.

I was inspired to create this checklist after reading The Checklist Manifesto by Atul Gawande. It highlights the important role of the humble checklist in several high-complexity industries like the Air Force, hospitals, disaster response, skyscraper construction and many other business activities.

Please note that this is by no means an exhaustive list. It's a list I've compiled over time based on experience and research. The cybersecurity landscape is always evolving, and I'd like to keep this list updated as new threats emerge. If you have any contributions to share, please do not hesitate to contact me via the channels above.

The list contains only brief explanations, and the assumption is that you're familiar with Laravel. Each of the items alone can be expanded to an in-depth discussion. I encourage you to research any items that seem unfamiliar and check out sources like Securing Laravel that have detailed explainers and examples of most of these recommendations.

A special thank you to Stephen Rees-Carter for reviewing this checklist. Stephen is very well known in the Laravel security community, and runs an excellent newsletter called Securing Laravel. Check out some of his talks where he exploits many of the vectors outlined in this document (much to the horror of the audience!).

This document is divided into three main sections:

- Essential – these are essential security tips and checks that should be applied to every application.
- Recommended – these checks aren’t strictly essential but are highly recommended to strengthen the security posture of your application.
- Printable Security Checklist – a printable checklist of the tips and checks mentioned in the sections above.

Thank you for reading, and I hope this checklist proves to be a valuable resource in your developer toolkit.

Essential

Prevent SQL injection

Make use of Eloquent or the query builder and avoid any unprepared or raw SQL statements where possible. Keep in mind PDO column names are not bound, never use user input for column names. Pay special attention to validation where injection is also possible.

Rate-limit relevant forms

Rate-limiting should be applied to all forms susceptible to abuse. This includes login, registration, contact form, etc. Adding CAPTCHA as another layer of defence is also recommended.

Secure password reset procedures

When checking an email address for a password reset, provide as little feedback as possible. The same message should be shown ("If you have an account with us, you will receive a password reset email shortly."), whether the email address was found or not. Also beware of timing attacks that may reveal if an email address does in fact exist.

A password should never be sent via email. Allow the user to set this via the website using a time-limited token.

Don't commit secrets to your repo

Usernames, passwords, API keys, and everything else sensitive so be added to your `.env` file and never committed to your repo. A tool like TruffleHog can be used to catch any secrets that have been leaked by accident.

Secure cookies

This ensures cookies are only sent if the application is served over a secure (HTTPS) connection. Control it via your `.env` to disable when developing locally if needed.

HttpOnly cookies

Set all cookies (or at the very least, the Laravel session cookie) that shouldn't be accessible in JavaScript as `HttpOnly`. This will help mitigate the risk of client-side scripts accessing the cookie (and potentially stealing the session cookie via XSS). This is enabled by default for Laravel sessions, and you can confirm this in the `session.php` config file.

Encrypt sensitive data

All sensitive data like PII (Personal Identifiable Information) should be encrypted where possible. Laravel has a very easy-to-use encryption service.

Prevent sensitive data exposure

Avoid exposing information about your app that can be used in an enumeration attack. If you're using auto-incrementing IDs for records, expose a corresponding UUID v4 (or similar scheme that can't be enumerated) instead.

Securely handle file uploads

Never trust user-uploaded files. Implement checks like file size, type, and limit the number of uploads (you can add this all to a custom validation rule). Auto-generate file names, and upload to non-public directories or secured 3rd parties like AWS S3. See more risk explainers on the OWASP website.

Beware of user-supplied XML

Try avoiding XML from unknown sources as you might be exposed to XML external entity (XXE) injection. Use the latest versions of PHP XML, and for older versions, set `libxml_disable_entity_loader` to `true` to prevent the ability to load external enties.

Avoid open redirects

Allowing users to tamper with redirect URLs (you've likely seen URLs like these before: `example.com/login?redirect=...`) could open you up to exploitation. Maintain an allow-list for redirects or keep them in a session if possible.

Check user access control

Users should only be allowed to perform their intended actions. For example, changing the order number in the browser URL bar shouldn't allow them to access the order for another user. And don't rely on hidden URLs (security through obscurity). Laravel has excellent authorisation functionality out the box.

Escape all user-supplied data

Use `{{ }}` instead of `{!! !!}` if using Blade for templates to prevent XSS (Cross-Site Scripting) attacks. If you need to allow for HTML, use a package like HTMLPurifier to reduce the chance of exploitation.

Avoid serialize() / unserialize()

Especially if you're using untrusted user input. Use `json_encode()` and `json_decode()` instead.

Validate all input

All input received from users should be validated to prevent unexpected values or size overflows. Laravel has an excellent validation system, plus you can create custom rules for most scenarios.

Log everything

There is a reason Security Logging and Monitoring Failures is in the OWASP Top 10. Keeping logs of everything from errors to login and password reset attempts would help in identifying the cause if a breach ever occurred. Setting up monitoring (e.g. via WAF or log alerts) will also provide you with early warnings that something suspicious may be happening.

Keep dependencies updated

Over time, dependencies may become vulnerable and need to be updated, especially if it’s a security-related patch. Remove all dependencies you don’t need, as this will reduce your attack surface.

Turn off debug mode for any public applications

Debug mode on production can expose your sensitive configuration values. If you have a website on a dev location and you need debug enabled, add basic auth so only authorised users can access the app.

Use SRI for external styles and scripts

SRI (Subresource Integrity) is essential for externally loaded styles and scripts. If the included styles/scripts are versioned and served from a CDN, it might be more secure to download and serve them from your application. You can use a service like SRI Hash Generator to generate an SRI hash for your loaded style/script if it’s not already provided. However, using SRI is not always possible, and sites like Stripe will require you to load their external script instead.

Use SSL/TLS certificates

Your entire production application should be served over HTTPS. Put redirects in place to ensure HTTP requests are redirected to HTTPS. Enable HSTS. Run your app through an SSL/TLS test to flag any related issues.

Recommended

Add a security.txt file

Place a `security.txt` file at `/.well-known/security.txt` containing the contact details a person can use to inform you if they find a security issue on your website. You can generate one at https://securitytxt.org/

Use separate encryption keys

If you’re allowing users to encrypt data or exposing raw input and encrypted results (encryption oracle), use a separate encryption key rather than the default Laravel `APP_KEY` for these operations.

Implement MFA functionality for your users

Allow your application users and/or admins to setup MFA. Most consumers expect this functionality these days. Use an authenticator-based solution rather than implementing an SMS/text message approach if possible.

Use the proper functions for randomness

If you’re generating any random data in your application (numbers, strings, etc.) be sure to use functions that produce cryptographically secure randomness. These include PHPs `random_int()` and Laravel’s `Str::random()` (which uses PHPs `random_bytes()`).

Implement security headers

Security headers being enabled on your server can add another layer of defence against things like XSS. Most of these are very easy to add. Run your website through securityheaders.com for a report and links to additional instructions.

Printable Security Checklist

Essential

☐	Prevent SQL injection
☐	Rate-limit relevant forms
☐	Secure password reset procedures
☐	Don’t commit secrets to your repo
☐	Secure cookies
☐	HttpOnly cookies
☐	Encrypt sensitive data
☐	Prevent sensitive data exposure
☐	Securely handle file uploads
☐	Beware of user-supplied XML
☐	Avoid open redirects
☐	Check user access control
☐	Escape all user-supplied data
☐	Avoid serialize() / unserialize()
☐	Validate all input
☐	Log everything
☐	Keep dependencies updated
☐	Turn off debug mode for any public applications
☐	Use SRI for external styles and scripts
☐	Use SSL/TLS certificates

Recommended

☐	Add a security.txt file
☐	Use separate encryption keys
☐	Implement MFA functionality for your users
☐	Use the proper functions for randomness
☐	Implement security headers

Laravel