Laravel security — OWASP risks

Authorization, validation, credentials, headers, SSRF, uploads, observation

TL;DR

Laravel gives you safe primitives; it does **not** decide who may open an invoice, which URL the server may fetch, or what the browser may load. Security is a set of **explicit boundaries tested at the same level as business behaviour**.

1. Authorize on the server

```
// policy — model actions; gate — permissions
final class UserPolicy {
    public function viewAny(User $user): bool { return $user->hasPermissionTo('users.view'); }
}

// the super-admin escape hatch in ONE reviewable place
Gate::before(fn (User $user): ?bool => $user->isSuperAdmin() ? true : null);

// Laravel 11+ — controller middleware via HasMiddleware (not $this->middleware())
final class AdminUserController extends Controller implements HasMiddleware {
    public static function middleware(): array {
        return ['auth', 'can:viewAny,'.User::class];
    }
}
```

> RULE Client-side hiding is UX, **never** authorization. Every sensitive route has a feature test for a *denied* user.

2. Validate, bind, encode

- A Form Request is the HTTP boundary. Use `$request->validated()`, not `$request->only()` — unvalidated fields must not reach mass assignment.
- Eloquent / query builder parameterize. Never concatenate input into SQL, shell commands or **dynamic column names**.
- Output escaped by default; sanitize user HTML deliberately, only if it's a product feature.

3. Credentials and sessions

- Passwords: `hashed` cast / `Hash::make()` — irreversible.
- API token secrets hashed at rest; show the plaintext **once**.
- Reversible integration credentials: `encrypted` cast; protect `APP_KEY`, rotate with overlapping values.
- Sanctum: **narrow abilities**, revocation after compromise, rate limit login and reset, MFA where the risk warrants it.
- **Never log** `Authorization` headers, cookies or reset URLs.

4. Browser and deployment boundaries

- Serve only `public/`, `APP_DEBUG=false`, HTTPS, secure cookies.

- Headers deliberately: CSP matched to your *actual* scripts, `X-Content-Type-Options: nosniff`, clickjacking protection, a strict referrer policy, HSTS once HTTPS is stable.
- CSP: start **report-only** → check violations → enforce. A copied permissive policy misses the point.
- `composer audit` + npm audit in CI; pin updates, remove unused packages. This reports known advisories, it is not a full review.

5. SSRF, uploads, external input

> **SSRF** An endpoint that fetches a user-supplied URL can reach internal infrastructure. Parse + allowlist hosts, resolve DNS defensively, **reject loopback / link-local / private ranges**, disable redirects, short timeouts, restrict egress at the network layer. A hostname regex is **not** a defense.

Uploads: validate MIME + size, keep files outside the public root, generate names server-side, scan high-risk files. Signed URLs = time-limited access, **not** a replacement for an authorization check.

6. Observe and rehearse

Log failed authentications and authorizations with **minimal** identifiers; report exceptions to the tracker with no secrets or PII in the payload. Queue listeners and webhooks must be **idempotent**. The work is done when a test proves the bad path is rejected and an operator can investigate it safely.

Production checklist

- Policies/Gates cover every sensitive action and have feature tests.
- Requests use validated data; input to the DB and shell is never concatenated.
- Tokens are scoped and hashed; secrets are encrypted or owned by the environment.
- HTTPS, debug settings, the public web root and headers — reviewed per environment.
- SSRF, upload and webhook boundaries have explicit tests and operational limits.