SCIĄGA LARAVEL · SECURITY

Bezpieczeństwo Laravel — ryzyka OWASP

dominik-dev.pl
pełny artykuł →

Autoryzacja, walidacja, poświadczenia, nagłówki, SSRF, uploady, obserwacja

TL;DR

Laravel daje bezpieczne prymitywy; **nie** zdecyduje, kto może otworzyć fakturę, który URL może pobrać serwer ani co wolno załadować przeglądarce. Bezpieczeństwo to zestaw **jawnych granic testowanych na tym samym poziomie co zachowanie biznesowe**.

1. Autoryzuj na serwerze

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

// furtka super-admina w JEDNYM miejscu do review
Gate::before(fn (User $user): ?bool => $user->isSuperAdmin() ? true : null);

// Laravel 11+ — middleware kontrolera przez HasMiddleware (nie $this->middleware())
final class AdminUserController extends Controller implements HasMiddleware {
    public static function middleware(): array {
        return ['auth', 'can:viewAny,'.User::class];
    }
}
```

REGUŁA Ukrywanie po stronie klienta to UX, **nigdy** autoryzacja. Każda wrażliwa trasa ma feature test użytkownika z *odmową*.

2. Waliduj, binduj, koduj

- Form Request = granica HTTP. Używaj `$request->validated()`, nie `$request->only()` — niezwalidowane pola nie mogą wejść do mass assignment.
- Eloquent / query builder parametryzują. Nigdy nie doklejaj wejścia do SQL, komend shella ani **dynamicznych nazw kolumn**.
- Wyjście escapowane domyślnie; HTML użytkownika sanityzuj świadomie, tylko jeśli to funkcja produktu.

3. Poświadczenia i sesje

- Hasła: cast `hashed` / `Hash::make()` — nieodwracalne.
- Sekrety tokenów API zahashowane w spoczynku; plaintext pokaż **raz**.
- Odwracalne poświadczenia integracji: `encrypted` cast; chroń `APP_KEY`, rotuj z nakładaniem wartości.
- Sanctum: **wąskie abilities**, unieważnianie po kompromitacji, rate limit na logowaniu i resecie, MFA gdzie ryzyko uzasadnia.
- **Nigdy nie loguj** nagłówków `Authorization`, cookies ani URL-i resetu.

4. Granice przeglądarki i wdrożenia

- Serwuj wyłącznie `public/`, `APP_DEBUG=false`, HTTPS, bezpieczne cookies.

- Nagłówki świadomie: CSP dopasowane do *rzeczywistych* skryptów, `X-Content-Type-Options: nosniff`, ochrona anty-clickjacking, ścisły referrer, HSTS po ustabilizowaniu HTTPS.
- CSP: zacznij **report-only** → sprawdź naruszenia → wymuś. Skopiowana pobłażliwa polityka mija się z celem.
- `composer audit` + audyt npm w CI; przypinaj aktualizacje, usuwaj nieużywane pakiety. To zgłasza znane advisories, nie jest pełnym review.

5. SSRF, uploady, wejście zewnętrzne

> **SSRF** Endpoint pobierający URL użytkownika może sięgnąć infrastruktury wewnętrznej. Parsuj + allowlistuj hosty, defensywnie rozwiązuj DNS, **odrzucaj loopback / link-local / prywatne zakresy**, wyłącz redirecty, krótkie timeouty, ogranicz egress na warstwie sieci. Sam regex hostname'a **nie** jest ochroną.

Uploady: waliduj MIME + rozmiar, trzymaj poza publicznym rootem, nazwy generuj po stronie serwera, skanuj pliki wysokiego ryzyka. Podpisane URL-e = dostęp czasowy, **nie** zamiast sprawdzenia autoryzacji.

6. Obserwuj i ćwicz

Loguj nieudane uwierzytelnienia i autoryzacje z **minimalnymi** identyfikatorami; wyjątki do trackera bez sekretów i PII w payloadzie. Listenery kolejek i webhooki **idempotentne**. Praca jest ukończona, gdy test dowodzi odrzucenia złej ścieżki, a operator potrafi ją bezpiecznie zbadać.

Lista produkcyjna

- Policies/Gates obejmują każdą wrażliwą akcję i mają feature testy.
- Requesty korzystają ze zwalidowanych danych; wejście do bazy i shella nigdy nie jest konkatenowane.
- Tokeny mają scope i są zahashowane; sekrety zaszyfrowane albo należą do środowiska.
- HTTPS, ustawienia debug, publiczny web root i nagłówki — przeglądane dla każdego środowiska.
- Granice SSRF, uploadów i webhooków mają jawne testy oraz limity operacyjne.
