// OWASP TOP 10:2025

Don’t Get Pwned.
Be Lazy.

A pragmatic walk through the ten most critical web app risks — and the laziest PHP fix for each.

---

php[tek] 2026 • 50 minutes • PHP 8.x

// THE LAZY DEVELOPER'S MANIFESTO

Lazy ≠ Careless.

Lazy means you make the computer do the work. Three rules:

1. **Don't Invent.** If a library exists, use it.
2. **Don't Think.** If a tool can check it, automate it.
3. **Don't Trust.** Assume everyone is trying to break your code.

// A01:2025

A:01

Broken Access Control

Still #1. Most common form: `?id=1` → `?id=2`

// THE HARD WAY

Security Through Obscurity

~~Hide the link in the UI~~

~~Don't expose the URL~~

~~Hope nobody changes ?id=~~

Miss one spot in one route → you're exposed.

// THE LAZY WAY

Framework Policies / Voters

One file per model. The only source of truth.

```php
// app/Policies/DocumentPolicy.php
public function view(User $user, Document $document): bool
{
    return $user->id === $document->user_id
        || $user->isAdmin();
}
```

Doesn't matter if they change the URL. Framework calls this. Blocked.

// A02:2025

A:02

Security Misconfiguration

↑ Up from #5 in 2021. Complexity keeps winning.

APP_DEBUG=true in prod?

That's a **gold mine** for attackers.

- Stack traces → file paths, class names, table names
- Environment variables on the error page
- I have seen `DB_PASSWORD` displayed on screen

// THE LAZY FIXES

Four One-Time Setups

1. **Secrets out of git.** `.env` in `.gitignore`. Push protection on.
2. **Errors off in prod.** `display_errors=Off`, log to file.
3. **Default creds gone.** No `/phpmyadmin`, no sample configs.
4. **Headers from a package.** `spatie/laravel-csp` / NelmioSecurityBundle.

php.ini — dev vs prod

```
; development
display_errors = On
error_reporting = E_ALL

; production
display_errors = Off
log_errors     = On
error_log      = /var/log/php/error.log
```

Errors go to a log file. **Never** to the browser.

// A03:2025

A:03

Software Supply Chain Failures

Renamed and expanded. The whole chain is a target now.

// LET ROBOTS WATCH YOUR DEPENDENCIES

One Command in CI

```
composer audit --format=json
```

Dependabot Renovate Snyk

Never remember to check. A bot should nag you.

Beyond Known CVEs

- **Typosquatting** — `symphony/console` ≠ `symfony/console`
- **Dependency confusion** — public package with your private name
- **Compromised maintainers** — `Shai-Hulud` npm worm, Sep '25 → May '26
- **Abandoned packages** — last commit 3 years ago, 200 issues

Pin registries. Commit lock files. `composer install --no-scripts` in CI.

// A04:2025

A:04

Cryptographic Failures

Hashing ≠ encoding. Both are not encryption.

STOP.

If you're typing one of these near "password" or "secret":

~~`md5()`~~

~~`sha1()`~~

~~`base64_encode()`~~

Your cat can decode base64. md5 was broken before most of us shipped code.

// THE LAZY WAY TO HASH PASSWORDS

Two Functions. That's It.

```php
// register
$hash = password_hash($plaintext, PASSWORD_DEFAULT);

// log in
if (password_verify($plaintext, $storedHash)) {
    // success
}
```

Salts. Algorithm upgrades. Constant-time compare. **All free.**

// ENCRYPT DATA YOU NEED TO READ BACK

Sodium — Bundled Since PHP 7.2

```php
$key   = sodium_crypto_secretbox_keygen();          // store securely
$nonce = random_bytes(SODIUM_CRYPTO_SECRETBOX_NONCEBYTES);

$ciphertext = sodium_crypto_secretbox($plaintext, $nonce, $key);
$plaintext  = sodium_crypto_secretbox_open(
    $ciphertext, $nonce, $key
);
```

Authenticated encryption. 256-bit. XSalsa20 + Poly1305. **You pick nothing.**

// A05:2025

A:05

Injection

25 years old. Still on the list. Still string concatenation.

Bind, Don’t Concatenate

```php
// HARD WAY (STILL INSECURE)
$u  = mysqli_real_escape_string(
    $conn, $_POST['username']
);
$sql = "SELECT * FROM users
        WHERE username = '" . $u . "'";
```

```php
// LAZY WAY
$stmt = $pdo->prepare(
    'SELECT * FROM users
     WHERE username = :u'
);
$stmt->execute([
    'u' => $_POST['username']
]);
```

Or even lazier: `User::where('username', $name)->first();`

Not Just SQL

- **Command injection** — avoid `exec/shell_exec/system`
- **LDAP injection** — parameterized filters
- **Template injection** — never let users author Twig/Blade

Never mix user data with executable syntax.

// A06:2025

A:06

Insecure Design

Not a bug. A defense you never built.

10,000 / sec

password attempts on your login form

You have `password_hash`. You have HTTPS. You have **no rate limit.**

// RATE LIMITING

One Line of Middleware

```php
// Laravel
Route::post('/login', [AuthController::class, 'login'])
    ->middleware('throttle:5,1');  // 5 attempts / minute
```

```php
// Symfony — RateLimiter component
$limiter = $factory->create($request->getClientIp());
if (! $limiter->consume()->isAccepted()) {
    throw new TooManyRequestsHttpException();
}
```

Threat modeling, lazy edition.

”What’s the worst thing a user could do with this feature?”

Five minutes of thinking beats five days of incident response.

// A07:2025

A:07

Authentication Failures

Renamed in 2025. Same category: impersonation.

Lock Down Sessions

```php
// Order matters: BEFORE session_start()
ini_set('session.cookie_httponly', '1');
ini_set('session.cookie_secure',   '1');
ini_set('session.cookie_samesite', 'Lax');

session_start();
session_regenerate_id(true); // after login, prevents fixation
```

If `session_start()` already ran, these `ini_set` calls do nothing.

// BREACHED PASSWORD DETECTION

Have I Been Pwned — k-anonymity

```php
$hash   = strtoupper(sha1($password));
$prefix = substr($hash, 0, 5);
$suffix = substr($hash, 5);

$resp = file_get_contents(
    "https://api.pwnedpasswords.com/range/{$prefix}"
);
foreach (preg_split('/\r?\n/', $resp) as $line) {
    [$candidate] = explode(':', $line, 2) + [''];
    if ($candidate === $suffix) return true;
}
return false;
```

Only the first 5 hash chars leave your server. Run on registration + change.

Multi-Factor Authentication

Don't write crypto. Wire up a library.

```
# Laravel
composer require pragmarx/google2fa-laravel

# Symfony
composer require scheb/2fa-bundle
```

QR codes, secret storage, code verification — all handled.

// A08:2025

A:08

Software or Data Integrity Failures

Two things: insecure deserialization + unsigned updates.

The `unserialize()` Bomb

```php
// 💣 Remote code execution waiting to happen
$data = unserialize($_COOKIE['user_prefs']);
```

- Instantiates arbitrary objects
- Triggers `__wakeup()` / `__destruct()`
- One of the most exploited PHP bugs in the wild

// THE EMBARRASSINGLY SIMPLE FIX

Use JSON.

```php
$cookie = json_encode(['theme' => 'dark', 'lang' => 'en']);
$prefs  = json_decode($cookie, true);
```

JSON cannot instantiate objects. JSON cannot run code. **It's data.**

```php
// if you MUST unserialize, allow-list:
$d = unserialize($payload, ['allowed_classes' => [Dto::class]]);
```

Pipeline Integrity

- **Lock files committed.** `composer install`, never `update` in CI.
- **SRI on CDN scripts.** Browser refuses tampered JavaScript.

```
<script src="https://cdn.example.com/lib.js"
        integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K..."
        crossorigin="anonymous"></script>
```

// A09:2025

A:09

Logging & Alerting Failures

"Monitoring" became "Alerting" in 2025. Dashboards aren't the job.

181

DAYS

average time to identify a data breach

Log Exactly Five Events

1. Failed logins (user, IP, ts)
2. Access denied (403s)
3. Input validation failures
4. Privilege changes (role grants)
5. Auth events (login/logout/MFA)

Never log passwords, card numbers, or PII. Log the event, not the payload.

The Lazy Monitoring Stack

- **Monolog** → JSON output
- **Centralized aggregator** → Papertrail / Logtail / CloudWatch
- **One alert rule** → "> 50 failed logins / 5 min / IP"

Do that and you're ahead of 80% of PHP apps.

// A10:2025 — NEW

A:10

Mishandling of Exceptional Conditions

New category for 2025. The big one: **failing open**.

Fail Closed, Not Open

// BAD — UNDEFINED STATE

```php
try {
    $allowed = $policy->check(
        $user, $resource
    );
} catch (Throwable $e) {
    // what is $allowed now?
}
if ($allowed) { /* … */ }
```

// GOOD — DENY BY DEFAULT

```php
$allowed = false;        // default: deny
try {
    $allowed = $policy->check(
        $user, $resource
    );
} catch (PolicyCheckException $e) {
    $logger->warning(/* … */);
}
```

Check Return Values

```php
$data = file_get_contents($path);
if ($data === false) {
    throw new RuntimeException("Could not read {$path}");
}
processData($data);
```

PHP functions that return `false` on failure will silently poison everything downstream.

PHPStan level 6+ catches most of these for you.

// HONORABLE MENTION

What happened to SSRF?

Off the Top 10 in 2025. Still real. Especially in cloud.

```php
// Resolve, don't regex.
$ip = gethostbyname(parse_url($url, PHP_URL_HOST) ?: '');
if (! filter_var($ip, FILTER_VALIDATE_IP,
    FILTER_FLAG_NO_PRIV_RANGE | FILTER_FLAG_NO_RES_RANGE)) {
    throw new InvalidArgumentException('Blocked range');
}
```

Pin the resolved IP through the request, or DNS rebinding eats your check.

// CHAPTER 11

The Lazy Toolchain

Let robots enforce most of this for you.

PHPStan — Levels 0 through 10

```
composer require --dev phpstan/phpstan
vendor/bin/phpstan analyse src --level=6   # min for security
# --level=10                               # if you can stomach it
```

- Type juggling — remember `'0' == false` is `true`
- Undefined variables that mask missing validation
- Unchecked return values (straight out of A10)

// MINIMAL GITHUB ACTIONS

Two Commands. Zero Daily Effort.

```yaml
security:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - run: composer install --no-dev
    - run: composer audit --format=json
    - run: vendor/bin/phpstan analyse src --level=6
```

Either fails → PR can't merge.

The OWASP Top 10:2025 — Lazy Cheat Sheet

#	RISK	LAZY FIX
A01	Broken Access Control	Policies / Voters — one file per model
A02	Security Misconfiguration	display_errors=Off, secrets package
A03	Software Supply Chain	composer audit + Dependabot
A04	Cryptographic Failures	password_hash + Sodium
A05	Injection	PDO / ORM — bind, don't concat
A06	Insecure Design	Rate limit + "what could go wrong?"
A07	Authentication Failures	Session flags, HIBP, MFA library
A08	Integrity Failures	json_encode, lock files, SRI
A09	Logging & Alerting	5 events, central, anomaly alert
A10	Mishandling Exceptions	Fail closed, don't leak, check returns

// IF YOU ONLY DO FIVE THINGS

This Week's TODO

1. Add `composer audit` to CI.
2. Add `phpstan analyse --level=6` next to it.
3. Set `session.cookie_*` flags in production.
4. Replace any `unserialize()` on user data with `json_decode()`.
5. Put `throttle` middleware on `/login`.

Stay Lazy.

Stay Secure.

The less code you write, the fewer bugs you ship.

Going Deeper

- **OWASP Top 10:2025** — `owasp.org/Top10`
- **OWASP PHP Cheat Sheet** — `cheatsheetseries.owasp.org`
- **PHP: The Right Way** — `phptherightway.com`
- **Paragon IE blog** — definitive PHP crypto/security
- **Troy Hunt** — auth security & HIBP

Thank you.

// QUESTIONS?

No hoodies required.