SkillNewton

OWASP Top 10 Security Checklist

Concrete developer checks for each OWASP Top 10:2021 risk, from broken access control to server-side request forgery.

> This checklist turns the OWASP Top 10:2021 list of web application security risks into practical checks developers can run during code review and before release. It is for PHP and web developers, testers and students. Each section is one OWASP category with specific things to verify in your code and configuration. The list is based on the 2021 edition; check owasp.org for any newer edition and map these checks to it.

How to use it

1. **Review one category at a time.** Go through each section against your application, focusing on features that handle logins, data and user input.
2. **Check code and configuration.** Many items are in server, framework or cloud settings, not just in code, so review both.
3. **Test, do not assume.** Try the attack safely on your own staging site, for example changing an ID in a URL, to prove a control works.
4. **Track and repeat.** Log each gap as a ticket, fix it, and repeat the checklist for every major release.

A01:2021 Broken Access Control

- [ ] Deny access by default
  Every route and API endpoint should require an explicit permission check, except truly public pages.
- [ ] Check authorisation on the server for every request
  Hiding a button in the interface is not access control.
- [ ] Check object ownership, not just login
  Changing `?invoice=1001` to `1002` must not show another user's invoice (an IDOR flaw).
- [ ] Protect admin areas by role
  Test admin URLs while logged in as a normal user and while logged out.
- [ ] Restrict CORS to trusted origins
  Never reflect any origin with credentials allowed.
- [ ] Disable directory listing and remove backup files from the web root
  Files such as `.git/`, `.env` and `.bak` copies must not be downloadable.
- [ ] Invalidate sessions on logout and expire them on the server
  Logging out should end the session, not just delete the cookie in the browser.

A02:2021 Cryptographic Failures

- [ ] Use HTTPS everywhere and enable HSTS
  Redirect HTTP to HTTPS and add `Strict-Transport-Security` once HTTPS is stable.
- [ ] Use TLS 1.2 or 1.3 only
  Disable SSL and TLS 1.0/1.1. Test your server with an online TLS checker.

- [ ] Hash passwords with a slow, salted algorithm
  In PHP use `password_hash()` (bcrypt or Argon2). Never MD5, SHA-1 or unsalted hashes.
- [ ] Classify sensitive data and do not store what you do not need
  Personal data, card details and tokens should be minimised, and encrypted at rest where stored.
- [ ] Use proven libraries for encryption
  In PHP use the Sodium extension; never write your own crypto.
- [ ] Keep keys and secrets out of source code
  Use environment variables or a secrets manager and rotate keys if they leak.
- [ ] Generate tokens with a secure random function
  Use `random_bytes()` or `random_int()`, not `rand()` or `mt_rand()`.

A03:2021 Injection

- [ ] Use prepared statements with bound parameters for all SQL
  Never build queries by joining strings with user input. PDO and MySQLi both support this.
- [ ] Escape output for its context to prevent XSS
  Use `htmlspecialchars($value, ENT_QUOTES, 'UTF-8')` for HTML, or your template engine's auto-escaping.
- [ ] Validate input with allow-lists
  Check type, length, format and range on the server, for example sort columns against a fixed list.
- [ ] Avoid passing user input to shell commands
  If unavoidable, use `escapeshellarg()` and a fixed command; prefer a library function instead.
- [ ] Add a Content Security Policy
  A CSP limits where scripts can load from and reduces the damage of an XSS bug.
- [ ] Watch for injection in other interpreters
  LDAP, XPath, NoSQL queries, email headers and template engines can all be injected.

A04:2021 Insecure Design

- [ ] Threat-model new features before building them
  Ask what could go wrong, who might abuse it and how you would detect it.
- [ ] Add rate limits to sensitive flows
  Login, password reset, OTP verification, coupon and contact forms need limits per user and per IP.
- [ ] Design business logic with abuse in mind
  For example, recheck prices and quantities on the server at checkout rather than trusting the browser.
- [ ] Use secure design patterns and shared components
  Reuse a tested authentication and authorisation module instead of rewriting it per feature.
- [ ] Write abuse cases alongside user stories
  "As an attacker, I try to reset another user's password" becomes a test case.

A05:2021 Security Misconfiguration

- [ ] Turn off debug mode and detailed errors in production
  Set `display_errors=Off` and use your framework's production environment.
- [ ] Remove default accounts, sample apps and unused features
  Change all default passwords and uninstall what you do not use.

- [ ] Set security headers
  Include Content-Security-Policy, X-Content-Type-Options: nosniff, Referrer-Policy and frame-ancestors (or X-Frame-Options).
- [ ] Harden cloud storage and services
  Check storage buckets are not public unless intended and that admin consoles are not exposed.
- [ ] Keep configuration the same across environments
  Automate server set-up so staging and production do not drift apart.
- [ ] Disable XML external entities if you parse XML
  XXE was merged into this category in 2021.

A06:2021 Vulnerable and Outdated Components

- [ ] Keep an inventory of all dependencies
  Include Composer and npm packages, CMS plugins, server software and their versions.
- [ ] Check for known vulnerabilities regularly
  Run `composer audit` and `npm audit`, or enable automated dependency alerts in your Git host.
- [ ] Run supported versions only
  Check the PHP, framework and database support schedules and plan upgrades before end of life.
- [ ] Remove unused packages and plugins
  Every unused component is still attack surface.
- [ ] Download components only from official sources
  Verify package names carefully to avoid typosquatted packages.

A07:2021 Identification and Authentication Failures

- [ ] Offer or require multi-factor authentication
  Prioritise MFA for admin and high-value accounts.
- [ ] Block weak and breached passwords
  Require a sensible minimum length and check against common and known-breached password lists rather than forcing complex rules.
- [ ] Limit and slow down failed login attempts
  Lock out or add delays after repeated failures, and alert on unusual patterns.
- [ ] Regenerate the session ID on login
  In PHP call `session_regenerate_id(true)` after successful authentication to prevent session fixation.
- [ ] Use generic login error messages
  "Invalid email or password" avoids revealing which accounts exist.
- [ ] Make password reset tokens single-use and short-lived
  Send them only to the registered email and expire them after use.

A08:2021 Software and Data Integrity Failures

- [ ] Do not unserialise untrusted data
  Avoid `unserialize()` on user input; use `json_decode()` instead.
- [ ] Use Subresource Integrity for third-party scripts
  Add an `integrity` attribute to scripts and styles loaded from CDNs.

- [ ] Commit and verify dependency lock files
  Keep `composer.lock` and `package-lock.json` in version control.
- [ ] Protect your CI/CD pipeline
  Restrict who can change build scripts and deploy, and keep pipeline secrets out of logs.
- [ ] Verify signatures on updates and downloads where available
  Do not auto-update from unverified sources.

A09:2021 Security Logging and Monitoring Failures

- [ ] Log logins, failed logins, access denials and admin actions
  Include time, user, IP address and action.
- [ ] Never log passwords, tokens or full card numbers
  Mask sensitive values before writing them to logs.
- [ ] Protect logs from tampering and injection
  Store them away from the web root, encode user input in log entries and back them up.
- [ ] Set alerts for suspicious activity
  For example many failed logins, a new admin account or a spike in errors.
- [ ] Keep logs long enough to investigate
  Agree a retention period that meets legal requirements; in India, CERT-In directions require some organisations to keep logs for 180 days.
- [ ] Have an incident response plan
  Know who to call, how to contain an incident and how to report it.

A10:2021 Server-Side Request Forgery (SSRF)

- [ ] Validate any URL the server fetches for a user
  Use an allow-list of permitted hosts and schemes (usually only https).
- [ ] Block requests to internal and metadata addresses
  Reject private ranges, `localhost`, `169.254.169.254` and similar, after resolving DNS.
- [ ] Do not follow redirects blindly
  A permitted URL can redirect to an internal address; disable redirects or re-check each hop.
- [ ] Do not return raw responses to the user
  Send back only the data you expect, not the full fetched content or error details.
- [ ] Segment the network
  Servers that fetch remote URLs should have limited access to internal systems.

Online version, with the matching free guide: skillnewton.com/resources/templates/owasp-security-checklist/
Learn this hands-on: Advanced PHP Web Development & Full-Stack Application Development – skillnewton.com/courses/advanced-php-course/