nautillo pro

Web Application Security Testing Checklist 2026

69 checks • mapped to the OWASP Top 10:2025 • updated November 17, 2026 • nautillo.pro/blog/security-testing-checklist

Work through each category before launch and after major releases. Fix **Critical** items first — attackers look for them in the first minutes of probing an application. Items marked **DAST** can be tested automatically from outside against your running app; **Review** items need a look at code, configuration or process. Test only applications you own or are authorized to test.

Critical fix before launch **High** direct impact **Medium** needs conditions **Low** hardening

DAST automatable · **Review** manual

Access Control

A01:2025 Broken Access Control

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Every endpoint checks that the logged-in user owns the record it returns or changes (no IDOR)	Log in as user A, request user B's object by changing the ID in the URL, body or query — expect 403/404	CRITICAL	DAST
☐	Admin-only functions are blocked for regular users	Call admin URLs and API routes with a normal user's session — expect 403	CRITICAL	DAST
☐	Authorization is decided on the server, never from client-supplied roles or flags	Tamper with role, isAdmin or plan fields in requests, cookies and tokens; the response must not change	CRITICAL	DAST
☐	Users cannot raise their own privileges by adding fields to update requests (mass assignment)	Add role, is_admin or owner_id to a profile update request and check whether it is stored	CRITICAL	DAST
☐	Deleted or deactivated users lose access immediately	Deactivate a test user, then replay one of their earlier requests and session	HIGH	DAST
☐	Server-side requests cannot be pointed at internal addresses (SSRF)	Submit URLs to 127.0.0.1, private ranges and 169.254.169.254 in URL-fetching features — they must be refused	CRITICAL	DAST
☐	Multi-tenant data is filtered by tenant on every query	Review data-access code for a tenant filter; test cross-tenant IDs with two test accounts	CRITICAL	Review

Security Misconfiguration

A02:2025 Security Misconfiguration

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Security headers set: Content-Security-Policy, X-Frame-Options or frame-ancestors, X-Content-Type-Options, Referrer-Policy	Inspect response headers on key pages (curl -I or browser dev tools)	MEDIUM	DAST
☐	HSTS enabled so browsers always use HTTPS	Check for a Strict-Transport-Security header with a long max-age	HIGH	DAST

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	CORS does not reflect arbitrary origins or allow '*' with credentials	Send a request with Origin: https://evil.example and check Access-Control-Allow-Origin / -Credentials	HIGH	DAST
☐	Debug mode, verbose error pages and framework consoles are off in production	Trigger errors (bad input, unknown routes) and look for stack traces, framework names or debug toolbars	MEDIUM	DAST
☐	Directory listing is disabled	Request folders such as /uploads/ or /static/ and check for an index of files	MEDIUM	DAST
☐	No exposed .git, .env, backup or config files	Request /.git/HEAD, /.env, /backup.zip, /config.json and similar predictable paths	CRITICAL	DAST
☐	Default credentials changed on every admin panel, database and service	Inventory admin interfaces and try vendor default logins in a test environment	CRITICAL	Review
☐	Admin and internal interfaces are not reachable from the public internet	Scan for /admin, management ports and internal dashboards from outside your network	CRITICAL	DAST
☐	Cloud storage buckets are private unless intentionally public	Review bucket policies; try anonymous listing of known bucket names	CRITICAL	Review
☐	Forgotten staging, preview and test environments are removed or protected	Enumerate subdomains (certificate transparency logs) and check each is still needed and protected	MEDIUM	Review

Software Supply Chain & Components

A03:2025 Software Supply Chain Failures

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	No end-of-life server software, frameworks or runtimes in production	Fingerprint versions from headers and responses; compare with vendor end-of-life dates	HIGH	DAST
☐	No dependencies with known exploited vulnerabilities (CISA KEV) or critical CVEs	Run a dependency audit (npm audit, pip-audit, Dependabot) in CI and fix or pin	HIGH	Review
☐	Third-party scripts loaded with Subresource Integrity (SRI) or from your own origin	Check <script> tags from CDNs for integrity= attributes	MEDIUM	DAST
☐	Lockfiles committed and builds reproducible	Confirm package-lock / poetry.lock / go.sum are in the repository and used in CI	MEDIUM	Review
☐	CI/CD secrets are scoped, rotated and never printed in logs	Review pipeline variables, permissions and recent build logs	HIGH	Review

Cryptography & Data Protection

A04:2025 Cryptographic Failures

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	TLS 1.2 or 1.3 only, with a valid certificate and strong ciphers	Run a TLS scan (e.g. testssl.sh) against every public hostname	CRITICAL	DAST
☐	No authenticated page or API is served over plain HTTP	Request http:// versions of login and API URLs — expect a redirect to HTTPS	CRITICAL	DAST

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Passwords hashed with Argon2, bcrypt or scrypt — never MD5, SHA-1 or plain SHA-256	Review the password storage code and a sample of stored hashes	CRITICAL	Review
☐	Secrets, API keys and connection strings are not in code, the repository or the frontend bundle	Run a secret scanner (gitleaks, trufflehog) on the repo and search built JavaScript for keys	CRITICAL	Review
☐	Sensitive data encrypted at rest where required (backups, personal data)	Review database, storage and backup encryption settings	HIGH	Review
☐	API responses return only the fields the client needs	Inspect list and detail API responses for hidden fields (emails, hashes, internal IDs)	HIGH	DAST

Injection

A05:2025 Injection

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	SQL queries use parameterized statements — no string concatenation	Send ' and " and time-based payloads to inputs and watch for errors or delays	CRITICAL	DAST
☐	NoSQL queries reject operator injection	Send {"$ne": null} or {"$gt": ""} in JSON fields used for lookups and login	CRITICAL	DAST
☐	User input is output-encoded in HTML, attributes and JavaScript (no XSS)	Submit a unique marker with <, >, " and ' to every input and check how it is reflected or stored	HIGH	DAST
☐	No DOM-based XSS from URL fragments, query parameters or postMessage	Review client code for innerHTML, document.write and eval fed by user-controlled sources	HIGH	Review
☐	OS commands never include user input without a strict allowlist	Send time-based payloads (; sleep 5) to inputs that may reach shell commands, in a test environment	CRITICAL	DAST
☐	Template engines never render user input as template code (no SSTI)	Submit {{7*7}}, ${7*7} and <%= 7*7 %> — the response must not show 49	CRITICAL	DAST
☐	XML parsers have external entities disabled (no XXE)	Send an XML body with an external entity reference to XML-accepting endpoints	HIGH	DAST
☐	Redirect targets are validated against an allowlist (no open redirect)	Set redirect / next / url parameters to an external domain and follow the response	MEDIUM	DAST

Insecure Design & Business Logic

A06:2025 Insecure Design

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Prices, quantities and discounts are calculated on the server	Change price, quantity or discount values in checkout requests — negative and zero values too	CRITICAL	DAST
☐	Multi-step workflows cannot be skipped or replayed	Call the final step of a flow (payment confirmation, approval) directly, or twice	HIGH	DAST
☐	Race conditions cannot duplicate one-time actions (coupons, credits, withdrawals)	Send the same request 10—20 times in parallel and compare the result with a single request	HIGH	DAST

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Expensive or sensitive actions are rate-limited (sign-up, password reset, exports, SMS)	Repeat the action rapidly and confirm throttling kicks in	MEDIUM	DAST
☐	Threat model reviewed for new features that handle money, permissions or personal data	Run a short abuse-case review before release: 'how could a user misuse this?'	MEDIUM	Review

Authentication & Sessions

A07:2025 Authentication Failures

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Login is rate-limited and protected against brute force and credential stuffing	Send many failed logins quickly and check for throttling, lockout or CAPTCHA	HIGH	DAST
☐	Login errors don't reveal whether an account exists	Compare responses and timing for a real vs a non-existent email	MEDIUM	DAST
☐	Session tokens are long, random and unpredictable	Collect several session IDs and check for patterns or short length	CRITICAL	DAST
☐	Session cookies use HttpOnly, Secure and SameSite	Inspect Set-Cookie headers after login	HIGH	DAST
☐	Logout and password change invalidate the session on the server	Reuse an old session token after logout or a password change — expect 401	HIGH	DAST
☐	Password reset tokens are single-use and expire within an hour	Reuse a reset link, and use one after expiry — both must fail	HIGH	DAST
☐	JWTs are verified for signature, algorithm and expiry (no alg:none, no weak secret)	Send a token with alg set to none, an expired token and a tampered payload — all must be rejected	CRITICAL	DAST
☐	MFA is available for admin and privileged accounts	Review admin account settings and enforce MFA for privileged roles	MEDIUM	Review
☐	OAuth / SSO redirect URIs are exact-match and state is validated	Try modified redirect_uri values and a missing or reused state parameter	HIGH	DAST

Software & Data Integrity

A08:2025 Software or Data Integrity Failures

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	File uploads validate type on the server and store files outside the web root	Upload a script file renamed to .jpg and a double-extension file, then try to request and execute them	CRITICAL	DAST
☐	Uploaded filenames are replaced or sanitized	Upload files named ../../evil.txt or with special characters and check where they land	HIGH	DAST
☐	No unsafe deserialization of user-controlled data	Review code for pickle, Java serialization, PHP unserialize or YAML load on request data	CRITICAL	Review
☐	Incoming webhooks verify a signature before acting	Send an unsigned or wrongly signed webhook request — it must be rejected	HIGH	DAST
☐	Deployments come only from the CI pipeline with reviewed code	Review who can deploy to production and whether branch protection is enforced	MEDIUM	Review

Logging & Alerting

A09:2025 Security Logging & Alerting Failures

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Logins, failed logins, permission changes and admin actions are logged	Perform these actions in staging and confirm a log entry with user, time and IP	MEDIUM	Review
☐	Alerts fire on suspicious patterns (brute force, mass data export, many 403s)	Simulate the pattern in staging and confirm someone is notified	MEDIUM	Review
☐	Logs never contain passwords, tokens, keys or full card numbers	Search recent application logs for secrets and personal data	HIGH	Review
☐	Logs are retained long enough to investigate an incident and protected from tampering	Review retention settings and who can delete or edit logs	LOW	Review

Error Handling & Exceptional Conditions

A10:2025 Mishandling of Exceptional Conditions

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Errors return generic messages — no stack traces, SQL errors or internal paths	Send malformed JSON, wrong types and oversized values and read the error bodies	MEDIUM	DAST
☐	Failures fail closed — an error in an auth or payment check denies the action	Review error branches in authorization and payment code; test with a dependency unavailable in staging	HIGH	Review
☐	Request size, upload size and timeouts are limited	Send very large bodies and slow requests and confirm they are rejected	MEDIUM	DAST
☐	Unhandled exceptions don't leave partial state (half-created orders, double charges)	Interrupt multi-step operations in staging and check data consistency	MEDIUM	Review

API Security

OWASP API Security Top 10

	CHECK	HOW TO TEST	SEVERITY	TYPE
☐	Every API endpoint requires authentication unless it is intentionally public	Replay API calls without a token and with another user's token	CRITICAL	DAST
☐	APIs are rate-limited per user and per IP	Send bursts of requests and confirm 429 responses	HIGH	DAST
☐	GraphQL introspection is disabled in production	Send an introspection query (__schema) to the GraphQL endpoint	MEDIUM	DAST
☐	GraphQL query depth, complexity and batching are limited	Send deeply nested and batched queries and check they are rejected	MEDIUM	DAST
☐	Old API versions and undocumented endpoints are retired	Compare live routes (e.g. /api/v1, /api/internal) with your API documentation	MEDIUM	DAST
☐	API keys are scoped, rotatable and never sent in URLs	Review key permissions and search logs and URLs for keys in query strings	HIGH	Review

Run the DAST checks automatically

Nautillo Pro attacks your web app and APIs like a real attacker and proves every confirmed finding with the exact request and response. Start free at nautillo.pro — passive checks and 1 active attack scan a month on the free plan, unlimited active testing on Professional.

© 2026 NØRTH HŪMAN OÜ — Nautillo Pro. Free to use and share with attribution to nautillo.pro. This checklist covers common, testable vulnerability classes; it does not replace a full security assessment.