WPStack

SECURITY ENGINEERING FIELD GUIDE

# WordPress REST API Security & Nonce Verification Matrix

A practical engineering standard for authenticating requests, authorizing actions, preventing CSRF, validating data, and testing WordPress REST routes.

**Built to be used**

Print it, annotate it in discovery, attach it to a brief, or use it as a release gate. Every checkbox should produce a decision, owner, or piece of evidence.

## What is inside

| Module | Purpose |
|---|---|
| 01 | Security model and trust boundaries |
| 02 | Route control matrix |
| 03 | Nonce decision and verification matrix |
| 04 | Input, output, secrets, and webhooks |
| 05 | Abuse resistance and operational controls |
| 06 | Verification plan and release evidence |

Edition: 2026.1 | Publisher: WPStack | wpstack.online

# 1. Use the right control for the right threat

Authentication establishes identity. Authorization decides whether that identity may perform an action. A nonce helps protect a logged-in browser workflow from cross-site request forgery; it is not authentication, authorization, or replay-proof access control.

| Control | Question answered | Never substitute it with |
|---|---|---|
| Authentication | Who or what is making the request? | A nonce or hidden URL |
| Authorization | May this actor perform this action on this object? | Being logged in |
| CSRF protection | Did a trusted browser context initiate the action? | Capability checks |
| Validation | Is the value allowed and structurally valid? | Sanitization alone |
| Escaping | Can this value be safely rendered in this output context? | Input cleaning alone |
| Abuse controls | Can valid access be used too quickly or too broadly? | Authentication alone |

**Core rule**

Every private route needs an explicit permission_callback. Check the narrowest capability and, when relevant, authorization for the specific object. Public routes should deliberately return true and expose only intended data.

## Trust-boundary inventory

| Caller | Credential | Data accessed | State changed | Trust level |
|---|---|---|---|---|
| wp-admin browser | Cookie + REST nonce | | | |
| External service | | | | |
| Public visitor | None | | | |
| Scheduled job | Server context | | | |

# 2. Route security control matrix

Complete one row per endpoint. Split routes when different operations need different capabilities or expose different fields.

| Route type | Authentication | Authorization | CSRF / replay | Abuse control |
|---|---|---|---|---|
| Public read | None | Intentional public permission | Not applicable | Pagination, caching, rate/size limits |
| Logged-in read | Cookie, app password, OAuth, etc. | Capability + object access | REST nonce for cookie auth | Field minimization, throttling |
| Logged-in write | Cookie or external credential | Capability + object access | REST nonce for cookie auth | Idempotency, rate and body limits |
| Machine integration | App password/OAuth/signed credential | Credential scope | Replay-resistant credential design | Quotas, rotation, monitoring |
| Inbound webhook | Signature or provider verification | Allowed event/account | Timestamp + replay store | Body limit, allowlist where useful |

## Per-route worksheet

| Namespace / route | Methods | Permission callback | Args schema | Owner |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# 3. Nonce verification matrix

WordPress REST cookie authentication commonly sends a nonce created for the `wp_rest` action in the `X-WP-Nonce` header. WordPress validates it as part of cookie-authenticated REST requests. Nonces expire and can be refreshed, so never treat them as a one-time secret.

| Scenario | Nonce? | Required controls | Decision |
|---|---|---|---|
| Logged-in wp-admin JavaScript changes settings | Yes: REST nonce | Cookie auth + capability + validation | Appropriate |
| Logged-in browser reads private records | Yes for cookie REST auth | Capability + object-level authorization | Appropriate |
| Public GET returning intentionally public data | No | Explicit public permission + data minimization | Do not add fake security |
| Mobile app or external client | No REST nonce | App password/OAuth/another supported auth method | Use real credentials |
| Inbound webhook | No REST nonce | Provider signature, timestamp, replay defense | Verify server-to-server request |
| Destructive admin action outside REST | Usually action-specific nonce | Capability + nonce + validated object | Use relevant WP nonce API |

## Nonce implementation checks

- [ ] Generate and send the expected nonce for the exact WordPress flow; do not invent a reusable static token.
- [ ] Do not place sensitive credentials in URLs, logs, analytics events, or HTML that untrusted users can read.
- [ ] Handle expired nonces by re-authenticating or retrieving a fresh nonce through a trusted flow.
- [ ] Always keep capability and object-level checks in permission_callback or the authorized service layer.
- [ ] Write negative tests for missing, invalid, expired, and wrong-user credentials.

# 4. Validate input and minimize output

Define route arguments with types, required flags, validation callbacks, and sanitization callbacks where appropriate. Validation decides whether a value is allowed; sanitization normalizes it; escaping happens when data is rendered.

| Data | Validate | Sanitize / normalize | Output control |
|---|---|---|---|
| Integer ID | Positive, exists, allowed object | absint | Return only authorized fields |
| Enum/status | Strict allowlist | sanitize_key if suitable | Map internal values deliberately |
| Text | Length and business rules | sanitize_text_field | Escape for HTML/attribute/URL context |
| Rich content | Allowed structure and capability | wp_kses with explicit policy | Render with matching policy |
| URL | Scheme, host, purpose | esc_url_raw for storage | esc_url for output |
| File | Type, size, extension, content | Use WordPress upload APIs | Private storage/access where required |

## Data exposure checklist

- [ ] Return the minimum fields needed by this client; avoid serializing full database rows or user objects.
- [ ] Do not expose hashes, reset tokens, API secrets, internal paths, stack traces, private metadata, or unnecessary personal data.
- [ ] Use consistent, non-sensitive error responses; log diagnostic detail only to protected server logs.
- [ ] Set pagination ceilings and reject unbounded filters, expensive sorting, and arbitrary meta queries.
- [ ] Review schema callbacks and embedded resources for fields that bypass the main response policy.

# 5. Secure integrations, secrets, and webhooks

Outbound credentials and inbound webhooks create a second trust boundary. A valid signature does not make the payload semantically safe, and HTTPS does not prove who created a webhook.

## Webhook verification sequence

- [ ] Read the raw request body exactly as required by the provider before parsing or normalizing it.
- [ ] Verify the provider signature with a constant-time comparison and the documented algorithm.
- [ ] Validate a timestamp or expiry window and reject stale messages.
- [ ] Store an event ID or idempotency key and reject already-processed events.
- [ ] Confirm the event belongs to the expected account, environment, resource, and event type.
- [ ] Acknowledge quickly; queue slow work; make processing safe to retry.
- [ ] Rotate secrets, separate staging/production credentials, and revoke immediately after suspected exposure.

## Secrets checklist

- [ ] Keep secrets out of source control, client-side scripts, support exports, database dumps shared externally, and error messages.
- [ ] Limit who can view or change credentials and record ownership and rotation dates.
- [ ] Do not log authorization headers, cookies, nonces, signed payloads, or full sensitive request bodies.
- [ ] Document behavior when a credential expires or a remote service is unavailable.

# 6. Abuse resistance and operational safety

| Risk | Design response | Evidence |
|---|---|---|
| Brute force / enumeration | Rate limits, generic errors, monitoring | Limit and alert tests |
| Large payload / resource exhaustion | Body, file, page, and batch ceilings | Boundary and load tests |
| Duplicate writes | Idempotency key and transactional guards | Repeated-request test |
| Expensive queries | Allowlisted filters, indexed access, caching | Query plan and p95 timing |
| Privilege change during job | Re-check authority at execution where required | Revoked-access test |
| Sensitive logging | Redaction and restricted retention | Log review |

## Operational controls

- [ ] Define request IDs, security-relevant audit events, redaction policy, alert thresholds, and retention.
- [ ] Fail closed when authorization or signature verification cannot complete.
- [ ] Use timeouts and bounded retries for remote calls; avoid retries on unsafe non-idempotent operations.
- [ ] Document credential revocation, incident triage, data exposure assessment, and customer communication ownership.
- [ ] Re-test controls after WordPress, PHP, authentication plugin, proxy, or infrastructure changes.

# 7. Verification and release evidence

A route is not secure because the happy path works. Security evidence must include negative authorization, malformed data, boundary conditions, replay attempts, and operational failures.

| Test family | Required cases | Result / evidence |
|---|---|---|
| Authentication | Missing, invalid, expired, revoked, wrong environment | |
| Authorization | Lower role, wrong owner, deleted object, changed capability | |
| CSRF / nonce | Missing, invalid, expired, different session/user | |
| Validation | Wrong type, unknown enum, oversized, malformed, duplicate | |
| Output | Field exposure, error leakage, escaping context | |
| Abuse | Burst, deep pagination, large body, repeated event | |
| Failure | Timeout, remote 5xx, queue failure, DB error, retry | |

## Release sign-off

- [ ] Every route has a documented owner, purpose, data classification, and explicit permission_callback.
- [ ] Threat review and negative tests cover each trust boundary.
- [ ] API documentation names authentication, required capability, parameters, errors, limits, and examples.
- [ ] Secrets, logs, alerts, rotation, incident response, and deprecation are owned.
- [ ] Security behavior has been retested in the intended proxy/CDN/hosting configuration.



Reference basis: official WordPress Developer Resources covering REST API authentication, route registration, permissions, nonces, capability checks, data validation, sanitization, and escaping. Nonces are not authorization. Requirements vary by authentication method and deployment; verify current official documentation before release.