



Code Security Review Report

Internal - Yoast SEO WordPress Plugin

Client	Internal (Sample Report)
Application Name	Yoast SEO (wordpress-seo , WordPress plugin)
Classification	Unclassified
Review date	2026-09-18
Report Version	v1
Report Author(s)	Intuitive IT

Purpose	3
Executive summary	3
Scope & methodology	4
Risk rating methodology	5
Summary of findings	6
Detailed findings	7
SEC-2 Hardcoded OAuth client secret for SEMrush integration	7
DATA-1 Inconsistent encryption of third-party integration tokens at rest	8
DATA-2 Settings export includes unencrypted third-party tokens in a plaintext blob	8
HARD-1 Static analysis (CodeQL) does not cover the PHP codebase	9
SEC-1 Real private key and certificate committed for a local dev tool	9
SEC-3 Private key retrievable from a live public remote branch	10
AUTH-1 Notification dismiss/restore action has no CSRF protection	11
AUTH-2 Shortcode-filter AJAX action checks a nonce but not a capability	11
NET-1 AI OAuth callback route has no explicit rate limiting	12
Positive observations	12
Recommendations & next steps	13
Appendix	13
Assumptions	13

Purpose

This document is a security and architecture review of the Yoast SEO WordPress plugin ([Yoast/wordpress-seo](#), public GitHub repository, commit [6530cf4084105a9201bd8eafeeca001605cde9f7](#)), prepared by Intuitive IT as an independent, code-only review of an open-source dependency. It records the findings of a systematic review covering data storage, secrets management, authentication & access control, network exposure, hardening, third-party integrations, and code quality, and provides prioritized recommendations.

Note on applicability: the review methodology used here is designed primarily for auditing AI-generated ("vibe-coded") applications built without a professional development process. Yoast SEO is a long-established, professionally maintained plugin with its own contribution process and private security-disclosure program. The checklist was applied as a general-purpose security lens; several sections (hosting/deployment ownership, infrastructure network topology) don't apply in the usual sense, since the plugin runs inside end-users' own WordPress installations rather than on infrastructure Yoast operates.

Executive summary

Yoast SEO is a mature, actively maintained plugin with real security process already in place: dependency vulnerability scanning runs on every push/PR and weekly, Dependabot is configured, most REST API and AJAX endpoints are properly gated behind WordPress capability checks, and a newer internal OAuth integration (MyYoast) encrypts stored tokens and uses timing-safe comparisons correctly. No critical or high-severity issues were found — nothing in this review is directly and immediately exploitable by an anonymous external attacker against a production site.

The findings that were identified cluster around a consistent theme: an older generation of third-party integration code (SEMrush, Wincher, IndexNow) predates a more careful security pattern the team has since adopted elsewhere in the codebase (seen in the newer MyYoast integration). That older code stores integration tokens unencrypted, includes a static OAuth client secret embedded in public source, and feeds those same unencrypted tokens into a plaintext settings-export feature. None of these are exploitable by an outside attacker without an additional foothold (database read access, or someone sharing an exported settings file), but they are real defense-in-depth gaps worth closing. A couple of minor, low-impact issues were also found in legacy AJAX handlers, plus one process gap: the project's static code-scanning tool (CodeQL) does not currently cover the PHP codebase, only JavaScript.

Findings at a glance:

Severity	Count
 Critical	0
 High	0
 Medium	4
 Low	5
 Informational	0

Most important next step: Rotate the hardcoded SEMrush OAuth client secret and move that integration to the PKCE pattern already used elsewhere in the codebase (SEC-2), and bring the older SEMrush/Wincher/IndexNow token storage up to the same encryption-at-rest standard as the newer MyYeast integration (DATA-1), since that also closes the related export-leakage gap (DATA-2).

Scope & methodology

In scope: The full `wordpress-seo` repository as checked out at commit `6530cf4084105a9201bd8eafeeca001605cde9f7` (static source code only) — PHP application code (`src/`, `admin/`, `inc/`, `lib/`), configuration, CI/CD workflow definitions, and full local git history.

Out of scope: Any live WordPress installation running the plugin; hosting/infrastructure of any specific site; the separate Yoast SEO Premium plugin (license activation, IndexNow, Zapier, and WordProof sending logic all live there, not in this repository); Yoast's own backend services (`my.yoast.com`, SEMrush's and Wincher's own servers); other remote branches beyond the one identified in SEC-3.

Access & limitations: This was a code-only review — no live system, admin console, or database access was available or used. Every finding below is based on static source and git history, not confirmed runtime behavior. Specific limitations:

- Full git history was searched with targeted patterns (private key headers, cloud-provider key prefixes, `client_secret`, add/remove of `.env` files) rather than a complete line-by-line read of history on a 7,000+ file, multi-year repository; this is noted as a practical constraint, not a claim of a fully clean history.
- The `packages/` JavaScript workspace (3,900+ files) was spot-checked rather than exhaustively scanned with the same secret-detection patterns.
- Whether SEMrush's own OAuth server imposes additional protections around the exposed client secret (SEC-2) could not be assessed — that lives entirely on SEMrush's side.

- Whether the historical private key in SEC-3 was ever deployed or used against a live system is unknown from source alone.

Methodology: This review followed the [ai-security-review](#) checklist methodology (v1.2.0), covering architecture, data storage, secrets management, authentication & access control, network exposure, audit logging, hardening, third-party integrations, and code quality.

Risk rating methodology

Findings are rated on a qualitative scale based on the realistic likelihood of exploitation and the business impact if exploited — not on how alarming a vulnerability class sounds in the abstract. For each Critical, High, or Medium finding, the detailed write-up below also states the likelihood and blast radius that justify its rating.

Severity	Definition
 Critical	Directly and immediately exploitable by an external, unauthenticated party to access, modify, or exfiltrate sensitive data, or to take over the application/infrastructure.
 High	Exploitable by an authenticated user beyond their intended privilege, or by an external party under realistic conditions (e.g. a leaked-but-not-yet-abused credential, a missing server-side check reachable from the client).
 Medium	A real weakness that requires a specific precondition to exploit (insider access, chaining with another issue, a narrow window of opportunity), or that materially weakens defense-in-depth without being independently exploitable today.
 Low	A deviation from best practice with limited realistic impact on its own — hardening/hygiene work.
 Informational	Not a vulnerability. Worth recording: a design tradeoff, an out-of-scope item, an unverified/assumption-based item, or context useful for future reviews.

Positive observations (see below) are genuine strengths, not findings, and are not rated on this scale.

Summary of findings

ID	Title	Severity	Area	Owner
SEC-2	Hardcoded "confidential" OAuth client secret for SEMrush shipped in public source	● Medium	Secret management	Code
DATA-1	SEMrush/Wincher/IndexNow tokens stored unencrypted at rest	● Medium	Data storage	Code
DATA-2	Plaintext settings export includes those same unencrypted third-party tokens	● Medium	Data storage	Code
HARD-1	CodeQL static analysis covers only JavaScript, not PHP (the majority of the codebase)	● Medium	Hardening / CI	Process decision
SEC-1	Real (expired, dev-only) private key and certificate committed to the repo	● Low	Secret management	Code
SEC-3	Real private key present in git history, still fetchable from a	● Low	Secret management	Code / repo admin

ID	Title	Severity	Area	Owner
	live public remote branch, tied to an abandoned/never-deployed feature			
AUTH-1	Notification dismiss/restore AJAX action has no CSRF check	● Low	Auth & access control	Code
AUTH-2	Shortcode-filter AJAX action checks a nonce but not a capability	● Low	Auth & access control	Code
NET-1	AI OAuth callback route is unauthenticated by design with no explicit rate limiting	● Low	Network exposure	Code

Detailed findings

SEC-2 Hardcoded OAuth client secret for SEMrush integration

Severity: ● Medium

What this means: A password-like credential that should stay secret is instead baked into the plugin's publicly readable code, so it isn't really secret at all.

Details: `src/config/semrush-client.php` hardcodes a static OAuth2 `clientSecret` for the SEMrush integration, present unchanged since 2020. Because this plugin ships as public PHP source (both on GitHub and inside every WordPress install that uses it), this value is trivially extractable by anyone — it provides no real confidentiality. Exploitability is partially bounded because SEMrush's OAuth server redirects only to SEMrush's own domain, but the secret being public still means Yoast's OAuth client identity to SEMrush cannot be treated as trusted the way a real confidential client secret would be. Notably, a newer integration in the same codebase (Wincher) already uses PKCE — a pattern

designed for exactly this situation (a public client that can't keep a secret) — with no static secret at all.

Evidence: `src/config/semrush-client.php:36` (value not reproduced here). Present since commit `d0f898a04a` (2020-10-06), unchanged through the reviewed commit.

Recommendation: Migrate the SEMrush integration to the PKCE flow already used by the Wincher integration, removing the static secret entirely. In parallel, ask SEMrush to rotate/reissue the client credentials, since the current one has been public since 2020.

DATA-1 Inconsistent encryption of third-party integration tokens at rest

Severity:  Medium

What this means: Login credentials for some connected third-party services (SEMrush, Wincher, IndexNow) are stored in the site's database without any extra protection, while a newer integration in the same plugin stores its equivalent tokens encrypted.

Details: SEMrush and Wincher OAuth tokens and the IndexNow API key are stored in plaintext inside the plugin's main WordPress option (`wpseo`). By contrast, the newer MyYoast OAuth client encrypts its tokens before storing them as WordPress options. This is a defense-in-depth gap in the older integrations, not an externally exploitable vulnerability by itself — reading it requires database-level access, which is already a high-trust position on any WordPress site.

Evidence:

`inc/options/class-wpseo-option-wpseo.php:38,66-75,92-95,438-441` (plaintext token option keys);
`src/myyoast-client/infrastructure/token/token-storage.php:71-97` and `user-token-storage.php` (contrasting encrypted pattern in the newer integration).

Recommendation: Apply the same `Encryption`-wrapped storage pattern used by the MyYoast client to the SEMrush, Wincher, and IndexNow token/key storage.

DATA-2 Settings export includes unencrypted third-party tokens in a plaintext blob

Severity:  Medium

What this means: When a site administrator uses the plugin's built-in "export settings" feature (commonly done to share configuration with a developer or support person), any connected SEMrush/Wincher/IndexNow credentials get swept into that plaintext export along with ordinary settings.

Details: The settings-export feature serializes every registered options group into a plaintext, INI-style block of text displayed on the page for the user to copy — it is not

encrypted and not a server-managed file download. Because the third-party tokens from DATA-1 live in the same options group that gets exported, they are included in this output. This is a realistic real-world exposure path: administrators routinely export and paste plugin settings when asking for support, without realizing embedded API tokens travel along with them.

Evidence: `admin/class-export.php:82-163` (export logic); same option keys as DATA-1.

Recommendation: Exclude the token/credential fields from the exported blob (or encrypt them within it), independent of fixing DATA-1 — the export code should never surface secrets even if the underlying option storage is fixed.

HARD-1 Static analysis (CodeQL) does not cover the PHP codebase

Severity:  Medium

What this means: The plugin's automated code-scanning security tool is only looking at the small JavaScript portion of the codebase, not the much larger PHP portion where most of the plugin's logic lives.

Details: `.github/workflows/codeql-analysis.yml` configures CodeQL to scan only the `javascript` language. It also only runs on pull requests that modify the workflow file itself, plus a weekly scheduled scan — it does not run CodeQL against ordinary PHP code changes on every PR. A separate workflow (`security.yml`) does run a dependency-vulnerability checker against `composer.lock` on every push/PR, which is valuable but answers a different question (known-vulnerable dependencies) than static analysis of the project's own PHP code (custom coding-pattern vulnerabilities).

Evidence: `.github/workflows/codeql-analysis.yml:23` (language list); trigger `paths` restriction in the same file.

Recommendation: Add a PHP CodeQL configuration (or an equivalent PHP static-analysis security scanner) running on every pull request, not just weekly.

SEC-1 Real private key and certificate committed for a local dev tool

Severity:  Low

What this means: A real cryptographic key pair was checked into the repository years ago to make a local development tool stop complaining — it's expired and only used to run a mock API server on a developer's own machine, but it's still a private key that shouldn't be in version control.

Details: `apps/components/privatekey.pem` and `apps/components/certificate.pem` are a real RSA key and matching self-signed certificate, committed in 2020, used only to run a local `stubby` mock-API dev server for a components demo app. The certificate's validity window expired in July 2020.

Evidence: `apps/components/privatekey.pem`, `apps/components/certificate.pem`; referenced in `apps/components/package.json:10`; added in commit `8aa2203b94`.

Recommendation: Remove both files from the repository, regenerate a throwaway key locally if the dev tooling still needs one, and add a `*.pem/*.key` rule to `.gitignore` so this doesn't recur.

SEC-3 Private key retrievable from a live public remote branch

Severity: ● Low (*downgraded from an initial Medium rating — see rationale below*)

What this means: A real cryptographic private key was committed during an abandoned experiment that never shipped, and although it was later deleted from the main line of development, it's still recoverable by anyone from a branch that's still published on GitHub.

Details: `src/oauth/private.key`, a real RSA private key, was added in one commit and deleted twelve days later, both part of an abandoned "Yeast as OAuth provider" prototype. Neither commit is reachable from `trunk`, but both are reachable from `origin/feature/oauth-server`, which is still present on the public GitHub repository — so the key remains fetchable via a plain `git fetch`.

Severity was initially rated Medium on the basis that real private key material is retrievable from a public repo. On reflection, the feature this key belonged to never merged to `trunk` and was removed from its own branch within two weeks of being added — there is no evidence it was ever built into a released version of the plugin or exposed on any live, internet-facing service. If the key was never deployed, there is no live system it can be used against today, which puts the realistic blast radius in the same range as SEC-1 (a real key, but with no exploitable target) rather than a credential actively protecting something in production. It remains a Low-severity hygiene finding rather than a Medium one: whether it was used briefly in a developer's own test environment during that two-week window cannot be ruled out from source history alone, and the branch itself should still be cleaned up regardless of impact.

Evidence: Commits `6da11f9324` (2022-05-03, added) and `4439353c6a` (2022-05-17, removed), reachable from `origin/feature/oauth-server`.

Recommendation: Delete the `feature/oauth-server` remote branch (or rewrite its history to remove the key). Rotating/revoking it is a good precaution but not urgent, given no evidence it was ever deployed.

AUTH-1 Notification dismiss/restore action has no CSRF protection

Severity: ● Low

What this means: A logged-in user could be tricked (via a malicious link or page) into dismissing or restoring one of their own Yoast admin notices without meaning to — a minor annoyance, not a data-security issue.

Details: One handler for the `wp_ajax_yoast_dismiss_notification/wp_ajax_yoast_restore_notification` actions performs no nonce check at all. A second, unrelated handler is also registered on the same action name and does check a nonce correctly — but WordPress runs every handler registered against a hook, so the correctly-checked handler does not stop the unchecked one from also executing.

Evidence: `admin/class-yoast-notifications.php:96-97,134-145,211-224` (unchecked handler); `admin/ajax.php:57` and `admin/class-yoast-notification-center.php:103-137` (the separately-registered, correctly nonce-checked handler on the same action name).

Recommendation: Add nonce verification to the handler in `class-yoast-notifications.php`, matching the pattern already used in `class-yoast-notification-center.php`.

AUTH-2 Shortcode-filter AJAX action checks a nonce but not a capability

Severity: ● Low

What this means: A backend feature that processes shortcodes for the post-editing screen relies only on the fact that a user has the right one-time token, not on an explicit check of what that user is allowed to do.

Details: The `wp_ajax_wpseo_filter_shortcodes` handler verifies a nonce but does not separately call `current_user_can()`. In practice this is scoped safely today because the nonce is only ever exposed on screens (post-edit metabox, taxonomy edit) that already require Contributor-level access or higher — but the endpoint's own code doesn't independently enforce that.

Evidence: `admin/ajax/class-shortcode-filter.php:19-53`.

Recommendation: Add an explicit `current_user_can()` check to the handler itself, so its security doesn't depend entirely on where the nonce happens to be exposed today.

NET-1 AI OAuth callback route has no explicit rate limiting

Severity: ● Low

What this means: A newer AI-feature login-callback endpoint that has to accept requests from an external service (and therefore can't require the visitor to already be logged in) relies on a hard-to-guess one-time code with no additional throttling if someone tries to guess it repeatedly.

Details: The callback route accepts a caller-supplied user ID plus a short-lived verifier value, and stores whatever access/refresh tokens are submitted once the verifier matches. The verifier is generated from a 10-character random password combined with an email address and hashed, valid for 5 minutes — a large enough guessing space that brute-forcing it in that window is impractical even at a high request rate, but the endpoint itself does not enforce any rate limit or lockout as a backstop.

Evidence: `src/ai/authorization/user-interface/callback-route.php:56`,
`refresh-callback-route.php:56`,
`src/ai/authorization/infrastructure/code-generator.php:22-25`,
`code-verifier-handler.php:18`.

Recommendation: Add a rate limit or attempt cap on this endpoint as defense-in-depth, even though the verifier's size makes practical brute force unlikely today.

Positive observations

- Dependency vulnerability scanning (`symfonycorp/security-checker-action` against `composer.lock`) runs on every push, every pull request, and on a weekly schedule, in addition to Dependabot being configured.
- Security-sensitive comparisons (OAuth state, OIDC issuer/nonce) consistently use `hash_equals()` rather than a plain `==/strcmp`.
- The newer MyYoast OAuth integration is a strong reference pattern: capability-gated routes, encrypted token storage, and a deliberately throttled status-refresh endpoint with upstream backoff handling.
- No `wp_ajax_nopriv_` handlers exist anywhere in the codebase — every AJAX action requires an authenticated WordPress session by construction.
- No CORS header broadening and no plaintext HTTP calls to real external services were found anywhere.
- The vast majority of REST API routes are correctly capability-gated (`manage_options` or plugin-specific capabilities), and the small number of intentionally open (`__return_true`) routes are all read-only, non-sensitive endpoints.
- `.github/CONTRIBUTING.md` and `AGENTS.md` are unusually detailed and explicit — covering architecture, dependency-injection wiring, coding standards, and a clear,

private security-disclosure process (yoast.com/security-program) rather than public issue reports.

Recommendations & next steps

1. Rotate the SEMrush OAuth client secret and migrate that integration to the PKCE flow already used by Wincher (SEC-2).
 2. Encrypt SEMrush/Wincher/IndexNow token storage to match the MyYoast pattern, and strip credential fields from the settings-export feature independent of that fix (DATA-1, DATA-2).
 3. Add PHP coverage to the CodeQL workflow and have it run on every PR, not just weekly (HARD-1).
 4. As routine hygiene: delete or rewrite the `feature/oauth-server` remote branch to remove the retrievable (but never-deployed) private key (SEC-3); remove the committed dev-only key/certificate pair and tighten `.gitignore` against `*.pem/*.key` (SEC-1).
 5. Fix the two minor AJAX gaps (AUTH-1, AUTH-2) and add rate limiting to the AI OAuth callback route (NET-1) as routine hardening work.
-

Appendix

Assumptions

- No live WordPress installation, hosting console, or database was available; all findings are based on static source code and full local git history as of the reviewed commit.
- The Premium plugin, MyYoast backend, and third-party services' own server-side behavior (SEMrush, Wincher) were not accessible and are assumed to behave as documented/typical for OAuth providers unless code evidence suggested otherwise.
- Report attribution (Client/Author fields) reflects that this was an independent, self-directed review of an open-source dependency rather than a formal client engagement.