WordPress Security Benchmark — DRAFT

Full Stack Hardening Guide

WordPress 6.x on Linux (Ubuntu/Debian) | Nginx or Apache | PHP 8.x | MySQL 8.x / MariaDB 10.x+

Dan Knauss | February 16, 2026

Overview

This document provides prescriptive guidance for establishing a secure configuration posture for WordPress 6.x running on a Linux server stack. This benchmark covers the full stack: the operating system firewall, web server (Nginx or Apache), PHP runtime, MySQL/MariaDB database, and the WordPress application layer.

This benchmark is intended for system administrators, security engineers, DevOps teams, and WordPress developers responsible for deploying and maintaining WordPress installations in enterprise environments.

Target Technology

- WordPress 6.x (latest stable release recommended)
- Ubuntu 22.04+ / Debian 12+ (or equivalent RHEL/CentOS)
- Nginx 1.24+ or Apache 2.4+
- PHP 8.2+ (8.3+ recommended for new deployments)
- MySQL 8.0+ or MariaDB 10.6+

Note on Containerization: While this benchmark assumes a traditional Linux stack, the principles apply equally to containerized environments (Docker, Kubernetes). Configurations should be injected via environment variables or secret management systems where possible.

Profile Definitions

Level	Description
Level 1	Essential security settings for any WordPress deployment with minimal impact on functionality. Forms a baseline every site should meet.
Level 2	Defense-in-depth settings for high-security environments. May restrict functionality, require additional tooling, or involve operational overhead.

Assessment Status: Automated (verifiable programmatically) or Manual (requires human judgment).

1. Web Server Configuration

1.1 Ensure TLS 1.2+ is enforced

Level 1 | Automated

Only TLS 1.2 and TLS 1.3 should be accepted. TLS 1.0 and 1.1 contain known vulnerabilities. All major browsers have dropped support for legacy protocols.

Audit (Nginx): Verify ssl_protocols contains only TLSv1.2 and TLSv1.3. **Audit (Apache):** Verify SSLProtocol shows 'all -SSLv3 -TLSv1 -TLSv1.1'.

Remediation (Nginx): `ssl_protocols TLSv1.2 TLSv1.3;` **Remediation (Apache):** `SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1`

Default: All protocols enabled. **Reference:** https://ssl-config.mozilla.org/

1.2 Ensure HTTP security headers are configured

Level 1 | Automated

The web server should send: Content-Security-Policy, X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security (HSTS), Referrer-Policy, and Permissions-Policy.

Impact: Overly restrictive CSP may break inline scripts or third-party integrations. For Level 2, aim to remove unsafe-inline by using nonces or hashes.

Remediation (Nginx): Add the six headers with `always` directive in the server block. **Remediation (Apache):** Use the Headers module.

Default: No security headers set by default.

1.3 Ensure server tokens and version information are hidden

Level 1 | Automated

The web server should not disclose version numbers or module information.

Remediation (Nginx): `server_tokens off;` **Remediation (Apache):** `ServerTokens Prod` and `ServerSignature Off`

Default: Version exposed.

1.4 Ensure direct PHP execution is blocked in upload directories

Level 1 | Automated

PHP execution must be disabled in wp-content/uploads/.

Impact: None. Legitimate WordPress operations never require PHP execution from uploads.

Remediation (Nginx): Deny all PHP in the uploads location block. **Remediation (Apache):** Add .htaccess with FilesMatch denying .php.

1.5 Ensure rate limiting is configured for all API surfaces

Level 1 | Automated

Apply rate limiting to wp-login.php, xmlrpc.php, and the REST API (/wp-json/).

Impact: Aggressive limits may lock out legitimate users. Configure appropriate burst allowances and allowlist exceptions.

Remediation (Nginx): Define limit_req_zone directives and apply to relevant locations.

2. PHP Configuration

2.1 Ensure expose_php is disabled

Level 1 | Automated

Set expose_php = Off in php.ini. Prevents PHP version disclosure in X-Powered-By headers.

2.2 Ensure display_errors is disabled in production

Level 1 | Automated

Set display_errors = Off and log_errors = On. Displayed errors can reveal file paths and database details.

2.3 Ensure dangerous PHP functions are disabled

Level 1 | Automated

Disable functions allowing arbitrary command execution: exec, passthru, shell_exec, system, proc_open, popen, curl_multi_exec, parse_ini_file, show_source, pcntl_exec.

Impact: Some plugins may require specific functions. Test before deploying. The eval() function cannot be disabled via disable_functions.

Level 2 Recommendation: Use Snuffleupagus to mitigate eval() and provide additional hardening.

2.4 Ensure open_basedir restricts file access

Level 2 | Automated

Restrict PHP file operations to the WordPress installation directory and required system paths.

Impact: Must include WordPress root, /tmp, and the PHP session directory.

2.5 Ensure PHP session security is configured

Level 1 | Automated

Set session.cookie_secure = 1, session.cookie_httponly = 1, session.cookie_samesite = Lax, session.use_strict_mode = 1, session.use_only_cookies = 1.

Default: Insecure defaults (cookie_secure = 0, cookie_httponly = 0).

3. Database Configuration

3.1 Ensure the WordPress database user has minimal privileges

Level 1 | Automated

Grant only SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, INDEX, and DROP on the WordPress database.

Impact: Some plugins may require CREATE TEMPORARY TABLES or LOCK TABLES. Add only when verified necessary.

3.2 Ensure the database is not accessible from external hosts

Level 1 | Automated

MySQL/MariaDB should listen only on localhost (127.0.0.1) or a Unix socket.

3.3 Ensure a non-default table prefix is used

Level 1 | Manual

Use a prefix other than the default wp_. Automated attack tools assume the default prefix.

Impact: Best done at installation time. Changing on an existing installation requires updating all table names and option/usermeta references.

3.4 Ensure database query logging is enabled

Level 2 | Automated

Enable at minimum slow_query_log for forensic analysis. General query logging incurs significant I/O overhead and should be used selectively.

4. WordPress Core Configuration

4.1 Ensure DISALLOW_FILE_MODS is set to true

Level 1 | Automated

Prevents all file modifications through the admin interface. If an attacker gains admin access, they cannot install malicious plugins or upload web shells. Updates should be handled through deployment pipelines.

Impact: Plugin and theme updates cannot be performed through the Dashboard. An alternative mechanism (wp-cli, CI/CD) is required.

4.2 Ensure FORCE_SSL_ADMIN is set to true

Level 1 | Automated

Forces all admin and login pages over HTTPS. Without this, session cookies could be transmitted over unencrypted HTTP.

4.3 Ensure WordPress debug mode is disabled in production

Level 1 | Automated

WP_DEBUG and WP_DEBUG_DISPLAY must be false. If WP_DEBUG_LOG is enabled, direct to a non-public path.

4.4 Ensure XML-RPC is disabled

Level 1 | Automated

Commonly exploited for brute-force amplification (system.multicall) and DDoS via pingbacks.

Impact: Will break Jetpack, older WordPress mobile app versions, and XML-RPC-dependent tools.

Remediation: Block at the web server level. Additionally, disable trackbacks and pingbacks in Settings.

4.5 Ensure automatic core updates are enabled

Level 1 | Automated

Minor releases contain only security fixes. Disabling them leaves the site vulnerable to known, publicly disclosed exploits.

4.6 Ensure unique authentication keys and salts are configured

Level 1 | Automated

All eight keys (AUTH_KEY, SECURE_AUTH_KEY, LOGGED_IN_KEY, NONCE_KEY, and their SALT counterparts) must be unique random values. Generate via the WordPress.org API.

4.7 Ensure wp-cron.php is replaced with a system cron job

Level 1 | Automated

The built-in pseudo-cron fires on page loads, making execution timing unpredictable and exposing an additional PHP endpoint.

Remediation: Set DISABLE_WP_CRON to true, add a system cron job with wp-cli, and block external access to wp-cron.php.

5. Authentication and Access Control

5.1 Ensure two-factor authentication is required for administrators

Level 1 | Manual

All Administrator accounts must have 2FA enabled using TOTP-based apps or hardware security keys (WebAuthn/FIDO2). Do not use SMS-based 2FA.

Impact: Requires a 2FA plugin. WordPress core does not include 2FA natively as of version 6.9.

5.2 Ensure the number of administrator accounts is minimized

Level 1 | Manual

Each admin account is a potential entry point. Compromising any single admin account grants full site control.

5.3 Ensure maximum session lifetime is enforced

Level 1 | Automated

Privileged accounts should have 8-24 hour session limits. Also enforce idle session timeouts, minimize "Remember Me," and purge sessions on role changes.

Default: 48 hours without Remember Me; 14 days with.

5.4 Ensure user enumeration is prevented

Level 1 | Automated

Restrict the REST API user endpoint and author archive URLs for unauthenticated requests.

5.5 Ensure reauthentication is required for privileged actions

Level 2 | Manual

Implement action-gated reauthentication (sudo mode) for: plugin/theme installation and deletion, user creation and role promotion, application password creation, critical configuration edits, data export, and core updates.

Default: WordPress requires password confirmation only for profile email/password changes.

5.6 Ensure unauthenticated REST API access is restricted

Level 2 | Automated

Restrict to authenticated users only, except for specific public endpoints.

Impact: Will break decoupled (headless) installations or plugins relying on unauthenticated REST access.

5.7 Ensure a strong password policy is enforced

Level 1 | Manual

Minimum 12 characters, check against breached password lists, enforce length and entropy (not arbitrary complexity).

Note: As of WordPress 6.8, bcrypt is the default hash. Argon2id is supported on compatible PHP environments.

5.8 Ensure user roles and capabilities are defined in code

Level 2 | Manual

Define roles in a must-use plugin or wp-config.php for version control, auditability, and resistance to database tampering.

6. File System Permissions

6.1 Ensure WordPress files are owned by a non-web-server user

Level 1 | Automated

Files should be owned by a system user, not the web server process user (www-data, nginx, apache). The web server should have read access only.

6.2 Ensure wp-config.php has restrictive permissions

Level 1 | Automated

File permissions of 600 or 640. Not readable by the web server user directly.

Default: 644 (world-readable) in many configurations.

6.3 Ensure wp-config.php is placed above the document root

Level 2 | Manual

Prevents direct HTTP access even if a web server misconfiguration exposes PHP source.

Impact: Some hosting environments may not support this configuration.

7. Logging and Monitoring

7.1 Ensure WordPress user activity logging is enabled

Level 1 | Manual

Record all user activity: logins, failed attempts, content changes, user account changes, plugin/theme changes, settings modifications. Retain logs per compliance requirements. Export to SIEM for Level 2.

7.2 Ensure file integrity monitoring is configured

Level 2 | Automated

Use wp core verify-checksums and wp plugin verify-checksums on a daily schedule. Alert on unexpected changes in wp-includes/, wp-admin/, and plugin directories.

7.3 Ensure server-level malware detection is configured

Level 2 | Manual

Deploy server-level malware scanning (Imunify360, Linux Malware Detect, ClamAV). Schedule daily scans. Enable real-time monitoring of wp-content/ where supported.

8. Supply Chain and Extension Management

8.1 Ensure all unused plugins and themes are removed

Level 1 | Automated

Delete deactivated plugins and non-active themes (except one default fallback). Deactivated plugins remain on the file system and may contain exploitable vulnerabilities.

8.2 Ensure all plugins and themes are from trusted sources

Level 1 | Manual

Install only from the WordPress.org repository or verified commercial vendors. Never use nulled (pirated) plugins or themes.

8.3 Ensure plugin and theme updates are applied promptly

Level 1 | Manual

Security updates within 72 hours. Critical updates immediately or virtual-patched within 24 hours. Use EPSS alongside CVSS to prioritize remediation.

8.4 Ensure a Software Bill of Materials (SBOM) is maintained

Level 2 | Manual

Document all components: core version, plugins, themes, third-party libraries, PHP version, web server, and database. Cross-reference against vulnerability databases on a regular schedule.

9. Web Application Firewall

9.1 Ensure Web Application Firewall is configured

Level 2 | Manual

Deploy ModSecurity with the OWASP Core Rule Set (server-level) or a cloud WAF (Cloudflare, Akamai, Sucuri). WordPress-specific exclusion rules are necessary for server-level WAFs.

Note: Enterprise WordPress deployments should treat WAF as a baseline requirement.

10. Backup and Recovery

10.1 Ensure backup and recovery procedures are implemented

Level 1 | Manual

Automated backups (files and database), stored offsite, encrypted, and tested quarterly. Retain 30 days of daily backups and 90 days of weekly backups. Document the recovery procedure. Prefer server-level backups over WordPress plugin-based backups.

11. AI and Generative AI Security

IBM found 13% of organizations experienced a breach involving an AI model or application, and 97% involved systems lacking proper access controls.

11.1 Ensure AI API keys are securely stored

Level 1 | Automated

Store API keys in wp-config.php constants or environment variables. Never expose in client-side code, version control, or the database. Rotate any previously exposed keys.

11.2 Ensure AI-generated content is sanitized

Level 1 | Manual

Treat all AI output as untrusted input. Apply wp_kses_post() before storing, esc_html() or esc_attr() before rendering, and $wpdb->prepare() before database queries.

11.3 Ensure AI tool usage is governed by policy

Level 2 | Manual

Maintain a policy covering approved tools, data classification for AI inputs, authentication requirements, and disclosure of AI-generated content. IBM found 63% of organizations lack AI governance policies.

12. Server Access and Network

12.1 Ensure SSH key-based authentication is enforced

Level 1 | Automated

Disable password-based SSH. Set PasswordAuthentication no, PubkeyAuthentication yes, PermitRootLogin no.

12.2 Ensure SFTP is used and FTP is disabled

Level 1 | Automated

No FTP server software should be installed. SFTP operates over the encrypted SSH channel.

12.3 Ensure a host-based firewall is configured

Level 1 | Automated

Restrict inbound traffic to ports 80, 443, and a non-standard SSH port. Default deny incoming.

12.4 Ensure per-site process isolation is configured

Level 2 | Manual

Each WordPress site should run under a dedicated system user with its own PHP-FPM pool.

13. Multisite Security

13.1 Ensure Super Admin accounts are minimized and audited

Level 1 | Manual

Super Admin grants unrestricted access across all sites in the network. Limit to the absolute minimum. Use per-site Administrator roles for day-to-day operations. Review quarterly.

13.2 Ensure network-activated plugins are reviewed for cross-site impact

Level 2 | Manual

A vulnerability in a network-activated plugin affects every site simultaneously. Reserve network activation for plugins that genuinely require network-wide operation.

Appendix A: Recommendation Summary

ID	Recommendation	Level	Assessment
1.1	Ensure TLS 1.2+ is enforced	L1	Automated
1.2	Ensure HTTP security headers are configured	L1	Automated
1.3	Ensure server tokens are hidden	L1	Automated
1.4	Ensure PHP execution is blocked in uploads	L1	Automated
1.5	Ensure rate limiting is configured for all APIs	L1	Automated
2.1	Ensure expose_php is disabled	L1	Automated
2.2	Ensure display_errors is disabled	L1	Automated
2.3	Ensure dangerous PHP functions are disabled	L1	Automated
2.4	Ensure open_basedir restricts file access	L2	Automated
2.5	Ensure PHP session security is configured	L1	Automated
3.1	Ensure DB user has minimal privileges	L1	Automated
3.2	Ensure DB is not externally accessible	L1	Automated
3.3	Ensure non-default table prefix is used	L1	Manual
3.4	Ensure database query logging is enabled	L2	Automated
4.1	Ensure DISALLOW_FILE_MODS is true	L1	Automated
4.2	Ensure FORCE_SSL_ADMIN is true	L1	Automated
4.3	Ensure debug mode is disabled	L1	Automated
4.4	Ensure XML-RPC is disabled	L1	Automated
4.5	Ensure automatic core updates are enabled	L1	Automated
4.6	Ensure unique auth keys and salts are configured	L1	Automated
4.7	Ensure wp-cron.php is replaced with system cron	L1	Automated
5.1	Ensure 2FA is required for administrators	L1	Manual
5.2	Ensure admin accounts are minimized	L1	Manual
5.3	Ensure max session lifetime is enforced	L1	Automated
5.4	Ensure user enumeration is prevented	L1	Automated
5.5	Ensure reauthentication for privileged actions	L2	Manual
5.6	Ensure unauthenticated REST API is restricted	L2	Automated

5.7	Ensure strong password policy is enforced	L1	Manual
5.8	Ensure roles and capabilities are defined in code	L2	Manual
6.1	Ensure files are owned by non-web-server user	L1	Automated
6.2	Ensure wp-config.php has restrictive permissions	L1	Automated
6.3	Ensure wp-config.php is above document root	L2	Manual
7.1	Ensure user activity logging is enabled	L1	Manual
7.2	Ensure file integrity monitoring is configured	L2	Automated
7.3	Ensure server-level malware detection is configured	L2	Manual
8.1	Ensure unused plugins and themes are removed	L1	Automated
8.2	Ensure extensions are from trusted sources	L1	Manual
8.3	Ensure plugin/theme updates are applied promptly	L1	Manual
8.4	Ensure a Software Bill of Materials is maintained	L2	Manual
9.1	Ensure Web Application Firewall is configured	L2	Manual
10.1	Ensure backup and recovery procedures are implemented	L1	Manual
11.1	Ensure AI API keys are securely stored	L1	Automated
11.2	Ensure AI-generated content is sanitized	L1	Manual
11.3	Ensure AI tool usage is governed by policy	L2	Manual
12.1	Ensure SSH key-based authentication is enforced	L1	Automated
12.2	Ensure SFTP is used and FTP is disabled	L1	Automated
12.3	Ensure a host-based firewall is configured	L1	Automated
12.4	Ensure per-site process isolation is configured	L2	Manual
13.1	Ensure Multisite Super Admin accounts are minimized	L1	Manual
13.2	Ensure network-activated plugins are reviewed	L2	Manual

Related Documents

- **WordPress Security Architecture and Hardening Guide** — Enterprise-focused security architecture.
- **WordPress Security Style Guide** — Principles, terminology, and formatting conventions.
- **WordPress Security White Paper** (WordPress.org, September 2025) — Official upstream document.

Format adapted from CIS Benchmarks. Technical guidance draws on the OWASP Top 10 (2025), NIST SP 800-63B, the Verizon DBIR (2025), and IBM's Cost of a Data Breach Report (2025).