Writing About Security, Vulnerability, and Open Source Software: A Style Guide for the WordPress Ecosystem

Principles, Practices, and Terminology for Clear, Honest, and Empowering Communication

Dan Knauss

Version 1.1 — April 13, 2026

Contents

Table of Contents

always fails harder in the end. It's devastating to brands, products, reputations, and careers. We see this happen time and again.

Maximizing security — and trust — in open source requires exposing all our work (warts and all) to everyone for review (or exploitation) by anyone.

3. Writing about Security and Vulnerabilities in WordPress

3.1 Lead with Solutions, Not Fear

Be as accurate as possible about threats, but orient your writing toward solutions. Security writing should inform and empower, not alarm or paralyze. The goal is to help people understand real risks and take effective, proportionate action.

> **Guiding Principle** Don't use fear, uncertainty, and doubt (FUD) to sell solutions. Dispel fear with knowledge. Demonstrate how reasonable levels of risk can be managed. Foster confidence in the tools, information, and relationships that empower WordPress users to protect their web properties.

When describing threats, always provide an actionable path forward. What knowledge, tools, or relationships will most effectively reduce the risk? Lead with that. When writing about incidents or breaches, extract the lesson and present it constructively. There is almost always an upside in what can be learned and improved.

3.2 Be Realistic and Proportionate

Avoid both minimizing and overstating threats. Don't tell open-ended stories of vague, dark unknowns. Be specific about what the risk is, who it affects, and what can be done about it. If a vulnerability is critical, say so clearly. If a risk is low for most users, say that too.

Remember that the vast majority of WordPress compromises trace back to a few recurring causes: weak or reused passwords, outdated plugins with known vulnerabilities, and neglected server environments. This is not new information for security professionals, but it is new for many WordPress users. Repeat these messages clearly and without condescension.

3.3 Place Responsibility Where It's Constructive

When assigning responsibility, make sure it serves a purpose. Responsibility should point toward action, not blame. Be clear about what is within the user's control, what

depends on their hosting provider, and what falls to the WordPress core team or plugin developers. Level with your audience, but don't push them away.

Recommended	Avoid
"This vulnerability affects sites running Plugin X versions prior to 3.2. Update immediately to version 3.2.1, which includes a patch."	"Your site could be hacked at any moment if you use Plugin X."
"Strong passwords and two-factor authentication significantly reduce the risk of unauthorized access."	"If you get hacked, it's your own fault for using weak passwords."

3.4 Make Security Accessible and Engaging

Write for the WordPress user who is still learning. Security topics can be dry, intimidating, or both. Work to make them interesting, practical, and empowering. Use clear examples, relatable scenarios, and plain language. If a concept requires technical depth, build up to it. Always define terms on first use.

3.5 Writing about AI, LLMs, and Automated Security Tools

AI-powered tools are increasingly used in WordPress security—for malware scanning, anomaly detection, code review, and vulnerability assessment. When writing about these tools, follow these principles:

- **Be specific about capabilities.** Describe what the tool does in concrete terms (e.g., "pattern matching against known malware signatures," "anomaly detection in file modification patterns") rather than vague claims. Avoid describing AI tools as "intelligent," "thinking," or "understanding"—they perform computation, not cognition.
- **Disclose AI-generated content.** When content, code samples, or analysis are produced with AI assistance, say so. Transparency supports trust.
- **Address AI-specific threat vectors.** AI introduces its own risks to the WordPress ecosystem. The Verizon DBIR (2025) found that AI-assisted malicious emails have doubled over two years; IBM's Cost of a Data Breach Report (2025) found 16% of breaches involved attackers using AI. When relevant, discuss:
  - **Prompt injection** — attacks that manipulate AI tools into performing unintended actions.

  - **Training-data poisoning** — attacks that corrupt AI models by inserting malicious data into training sets.
  - **AI-generated phishing** — increasingly sophisticated social engineering content produced by LLMs (37% of AI-driven breaches per IBM 2025).
  - **Deepfake-based social engineering** — synthetic audio or video impersonating trusted individuals (35% of AI-driven breaches per IBM 2025).
  - **Shadow AI** — unsanctioned employee use of AI tools that may leak sensitive data or bypass security controls. See glossary.
  - **Insecure AI-generated code** — code produced by LLMs that introduces vulnerabilities (e.g., missing input sanitization, improper use of `$wpdb->prepare()`).

- **Avoid hype and anthropomorphism.** AI is a tool, not a colleague. Don't attribute agency, judgment, or intent to automated systems.

Recommended	Avoid
"The scanner uses pattern matching to identify known malware signatures."	"Our AI intelligently detects all threats."
"This code was generated with AI assistance and has been reviewed for security."	"Our AI wrote secure code."

3.6 Writing about Compliance and Regulatory Frameworks

WordPress is often deployed in environments subject to regulatory requirements (SOC 2, PCI DSS, HIPAA, GDPR, FedRAMP). When writing about compliance:

- **Don't claim WordPress is "compliant."** Software is not compliant; deployments are. A specific WordPress installation, configured and operated in a particular way, may meet the requirements of a given framework. The software alone does not.
- **Reference the specific framework and control.** Vague claims like "meets industry standards" are unhelpful. When making a compliance-related statement, cite the framework, the control or requirement, and how the WordPress configuration addresses it (e.g., "Enforcing 2FA for all administrator accounts supports NIST SP 800-53 IA-2(1)").
- **Acknowledge shared responsibility.** Compliance in a WordPress deployment depends on the software, the hosting environment, and the site operator. Be clear about which layer is responsible for what.

- **Distinguish between certification and alignment.** An organization can be *certified* against a framework (e.g., SOC 2 Type II) or *aligned* with its principles without formal certification. Use the correct term.

3.7 Context-Dependent Technical Recommendations

Many security recommendations — file permissions, server configuration values, timeout thresholds, rate limits — are highly dependent on deployment context. There are few universal "right answers," but there are universally wrong ones. When writing about context-dependent topics:

- **Present principles, not prescriptions.** Explain the security objective (e.g., "restrict file access to the minimum necessary") rather than prescribing a single value (e.g., "set permissions to 600") that may be incorrect for some environments.
- **Acknowledge variability.** Different hosting environments, deployment methods, and operational requirements produce different correct configurations. A containerized deployment, a shared hosting environment, and a dedicated server each have different filesystem ownership models.
- **Identify the universally wrong choices.** While the right answer varies, some answers are always wrong (e.g., 777 permissions, `GRANT ALL PRIVILEGES` on a production database). Call these out explicitly.
- **Provide context for specific values.** When you must cite a specific value, qualify it: explain the assumptions (e.g., "assumes the web server runs as `www-data` and is the only process that needs read access"), note when alternatives are appropriate, and cross-reference the Benchmark for auditable guidance.

Recommended	Avoid
"Restrict `wp-config.php` to the minimum permissions your deployment requires. Owner-read-only (`400` or `440`) is preferred when your deployment pipeline does not need write access."	"Set `wp-config.php` to `600`."
"Grant only the eight specific database privileges WordPress requires (SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, INDEX, DROP)."	"Set up the database and make sure the user has access."

4. Audience, Voice, and Tone

4.1 Know Your Audiences

Security writing in the WordPress ecosystem addresses several overlapping audiences. Your content may reach any combination of them, so clarity and accessibility are essential:

- WordPress site owners and administrators who manage one or more sites and need practical, actionable guidance.
- Developers and engineers building custom themes, plugins, or integrations who need technically precise information.
- Enterprise decision-makers (CTOs, CISOs, IT directors) evaluating WordPress for compliance, risk, and organizational security requirements.
- Non-technical stakeholders (marketers, content editors, business owners) who interact with WordPress daily but have limited security knowledge.
- WordPress community members and contributors, including those involved in core development, plugin review, and support forums.

4.2 Voice

Your voice should convey the brand's personality and values. It is:

- **Confident** — grounded in knowledge and experience, never bluffing or overpromising.
- **Candid** — honest about problems, limitations, and uncertainty.
- **Expert** — technically accurate, well-sourced, and current.
- **Accessible** — warm, clear, and human. Real people write this, and real people read it.
- **Open** — reflecting the open-source values of transparency, collaboration, and shared responsibility.

4.3 Tone

Tone adapts to context while the voice remains consistent. The default tone for security writing is:

- **Realistic about problems** — acknowledge risks squarely without catastrophizing.

- **Optimistic and Reassuring** — emphasize what can be done and what has been fixed. Reassure the audience that our team is on top of the problem.
- **Down-to-earth** — avoid jargon-heavy abstractions. Prefer plain language.
- **Clear, Concise, and Honest** — provide straightforward information without alarming the reader.

> **Tone Shift by Context** Vulnerability disclosure: measured, precise, actionable. No editorializing. Educational content: encouraging, patient, building understanding step by step. Incident response guidance: calm, clear, directive. Prioritize the most important actions first. Thought leadership: reflective, well-sourced, open to nuance and debate.

5. Inclusive Communication

5.1 Bring Outsiders In

Security writing can easily become a closed conversation among experts. Actively work against this. Explain jargon and technical terms the first time you use them. Spell out acronyms on first use. Helping people learn the vocabulary means helping them enter the community and take responsibility for their own security.

5.2 Language Choices

Use inclusive, contemporary language. Some traditional security terminology carries exclusionary connotations or has clearer modern alternatives:

Recommended	Avoid
allowlist / denylist	whitelist / blacklist
primary / replica	master / slave
brute-force attack / credential stuffing attack	brute-force hacking
threat actor	hacker (when meaning attacker)

Note: “brute-force attack” and “credential stuffing attack” are distinct categories. Use whichever term accurately describes the attack being discussed. See the glossary in Section 8 for definitions.

When referring to people who exploit systems maliciously, prefer specific terms like "threat actor," "attacker," or "cybercriminal" over the ambiguous "hacker," which has positive connotations in many technical communities.

5.3 WordPress-Specific Terminology

WordPress has its own vocabulary. Use terms consistently and prefer the forms familiar to the WordPress community:

- **Dashboard** — the WordPress admin interface (avoid "backend" in user-facing writing).
- **Plugin** — an extension that adds functionality to WordPress. Always one word, lowercase in running text.
- **Theme** — a collection of templates and stylesheets that control a site's visual presentation.
- **wp-admin** — the URL path to the WordPress Dashboard. Set in monospace (`wp-admin`) when referring to the path.
- **wp-config.php** — the primary WordPress configuration file. Always in monospace (`wp-config.php`).
- **Multisite** — a WordPress feature enabling multiple sites on one installation. One word, capitalized.
- **Auto-update** — WordPress's built-in mechanism for applying updates automatically. Hyphenated.

6. Technical Formatting Guidelines

6.1 Two-Font System

Use two font treatments to distinguish between human-readable and machine-readable terms:

- **Normal font** (the document's body typeface) for names of products, organizations, document titles, and human-facing concepts: WordPress, Cloudflare, an SSL certificate.
- **Monospace font** (like this) for code, commands, file paths, configuration values, and machine-facing identifiers: `wp-config.php`, `DISALLOW_FILE_MODS`, `wp_kses()`.

6.2 When to Use Monospace

- File names and paths: `wp-config.php`, `/wp-content/uploads/`
- Configuration constants and PHP functions: `FORCE_SSL_ADMIN`, `current_user_can()`
- Command-line tools and commands: `wp-cli`, `ssh`, `fail2ban`
- Database fields, table names, and environment variables
- HTTP headers, status codes, and URL parameters: `X-Frame-Options`, `403`, `?author=1`
- CVE identifiers and version numbers in technical context: `CVE-2024-1234`, `WordPress 7.0.1`

> **Boundary rule:** Use monospace when the version number is the point of the sentence—the reader needs to act on it (e.g., "Update to `3.2.1`"). Use normal font when the version appears as background context in running prose (e.g., "WordPress 7.0 introduced…").

6.3 When to Use Normal Font

- Product names and their versions in running prose: WordPress 7.0, Cloudflare, Patchstack.
- Names of organizations, teams, and conferences: OWASP, DEF CON, WordPress Security Team.
- Concepts and roles: administrator, two-factor authentication, session cookie.
- Error messages and security prompts when quoting them for a non-technical audience (use quotation marks).

6.4 Bold and Emphasis

Use bold sparingly for key terms being defined, important warnings, and UI elements the reader needs to find and click. Use italics for emphasis, titles of publications, and introducing new terms in running text.

6.5 Acronyms

Spell out acronyms on first use, followed by the abbreviation in parentheses: cross-site scripting (XSS), two-factor authentication (2FA). Use the acronym alone in subsequent references. If the acronym is more widely recognized than the full form (e.g., SQL, HTML, SSH), the spelled-out version can be omitted for technical audiences.

7. Writing about Vulnerabilities

7.1 Disclosure and Reporting

When writing about specific vulnerabilities, follow established responsible disclosure conventions:

1. Use the official CVE identifier when available.
2. Name the affected software and specific versions.
3. Describe the vulnerability type using standard terminology (XSS, SQL injection, CSRF, etc.).
4. State the severity using an established framework (CVSS score, Patchstack's severity rating, or the plugin repository's classification).
5. Provide the remediation: update to version X, apply a configuration change, or remove the affected component.
6. Credit the researcher or security team that reported the issue, consistent with responsible disclosure norms.

> **Template: Vulnerability Summary (Authenticated)** [Plugin/Theme Name] versions `[X]` through `[Y]` contain a [vulnerability type] vulnerability (`[CVE-YYYY-NNNNN]`) rated [severity]. Authenticated users with [role] access or above can [exploit description]. Update to version `[Z]` or later to resolve this issue. [Credit to researcher/team for responsible disclosure.]
>
> **Template: Vulnerability Summary (Unauthenticated)** [Plugin/Theme Name] versions `[X]` through `[Y]` contain a [vulnerability type] vulnerability (`[CVE-YYYY-NNNNN]`) rated [severity]. Unauthenticated attackers can [exploit description] without any login credentials. Update to version `[Z]` or later to resolve this issue. [Credit to researcher/team for responsible disclosure.]

7.2 Severity Language

Match the urgency of your language to the actual severity of the vulnerability. We use the Common Vulnerability Scoring System (CVSS) to assess which level of severity applies. Note: CVSS 4.0 was published in November 2023 and is the current standard; many vulnerability databases still report CVSS 3.1 scores alongside 4.0 during the transition. Use whichever version your source provides and note the version number (e.g., "CVSS 3.1: 8.8" or "CVSS 4.0: 8.7"). Reporting requirements vary based on this assessment:

- **Critical Severity** — Very serious vulnerabilities that could compromise a website detrimentally. Always reported to customers in a dedicated email.

- **High Severity** — Serious vulnerabilities requiring prompt action. Always reported to customers in a dedicated email.
- **Medium Severity** — Vulnerabilities with moderate impact. Development, Support, and Marketing decide whether to dedicate an email.
- **Low Severity** — Localized or low-impact issues. Reported in Product Update emails only.

CVSS Range	Internal Label	Default Communication Channel
9.0–10.0	Critical	Dedicated customer email
7.0–8.9	High	Dedicated customer email
4.0–6.9	Medium	Case-by-case decision by Development, Support, and Marketing
0.1–3.9	Low	Product update email

Recommended	Avoid
“Critical: unauthenticated remote code execution”	“Extremely dangerous flaw found in popular plugin!”
“Low severity: authenticated stored XSS requiring administrator role”	“Minor issue, probably nothing to worry about.”

While these severity levels mean different things for our customers, all vulnerability communications should follow the same professional procedures.

7.3 Avoiding Sensationalism

Security news attracts clicks, and sensationalism is common in the WordPress security ecosystem. Resist it. Inflated threat descriptions erode trust, cause unnecessary alarm, and make it harder for users to distinguish genuinely urgent issues from routine maintenance.

Write with precision. If a vulnerability requires administrator-level authentication to exploit, say so. If it affects a plugin with 200 active installations, provide that context. Help the reader assess whether the issue is relevant to them.

When writing about a vulnerability, include these context signals so readers can self-assess relevance:

- **Active install count** of the affected plugin or theme.
- **Authentication requirement** — does exploitation require a logged-in user, and at what role level?
- **Default vs. non-default configuration** — is the vulnerable feature enabled by default?
- **Affected version range** — which versions are vulnerable, and how far back does it go?
- **Auto-update availability** — can users receive the patch automatically?
- **EPSS score** — the Exploit Prediction Scoring System probability, when available. EPSS estimates the likelihood a vulnerability will be exploited in the wild within 30 days and is increasingly reported by databases like Patchstack alongside CVSS. Include EPSS as a supplemental data point (e.g., "EPSS: 0.04%" or "EPSS: 87%") to help readers gauge real-world urgency beyond theoretical severity.

7.4 Writing about Core vs. Plugin vs. Theme Vulnerabilities

WordPress core vulnerabilities, plugin vulnerabilities, and theme vulnerabilities have different disclosure norms, timelines, and audience expectations. Adjust your writing accordingly:

- **Core vulnerabilities** are handled by the WordPress Security Team, coordinated with major hosting providers, and typically auto-patched to all supported versions. The audience expects measured, factual language. Credit the Security Team's process.
- **Plugin vulnerabilities** are handled by individual plugin authors. Quality and response times vary widely. The WordPress Plugin Security Team on WordPress.org may force-update or close plugins. Emphasize the user's responsibility to update promptly.
- **Theme vulnerabilities** follow a similar pattern to plugins but receive less public attention. Provide the same level of specificity—name the theme, the affected versions, and the fix.

7.5 Naming Plugins and Themes in Vulnerability Writing

Always name the affected plugin or theme. Users cannot act on vague warnings. However, exercise proportionality:

- For widely used plugins (100,000+ active installations), the public interest in disclosure is high and the name will appear in vulnerability databases regardless.
- For smaller plugins (under 1,000 active installations), the same public disclosure reaches a much smaller affected audience but may disproportionately affect the

plugin author's reputation. Ensure the description is precise and fair—state facts, not judgments about code quality.
- Never editorialize about a plugin author's competence or responsiveness. Stick to what happened, what was fixed, and what users should do.

7.6 Operational Policy Boundary

This style guide defines *writing standards*—how to communicate about vulnerabilities clearly, accurately, and consistently. The operational procedures for *who does what and when* during a vulnerability response are maintained separately in §9 (Operational Appendix). This separation ensures the style guidance remains stable even as internal workflows evolve.

7.7 Writing about Supply Chain Incidents

Supply chain attacks targeting the WordPress ecosystem—compromised plugins, theme ownership transfers, dependency confusion, and hijacked developer accounts—are a growing concern. When writing about these incidents, apply specific framing:

- **Name the mechanism.** Be precise about how the compromise occurred: a plugin sale to a malicious buyer, a compromised developer account, a backdoored dependency, or a rogue commit in a build pipeline. Each has different implications for the affected user.
- **Distinguish intent from negligence.** A plugin that was deliberately backdoored after an ownership transfer is a fundamentally different event from a plugin that inadvertently included a vulnerable dependency. The writing should reflect the difference without speculating beyond known facts.
- **Provide a timeline.** State the compromise window—when the malicious code was introduced and when it was detected or removed. Users need this to assess whether their sites were exposed.
- **Describe the blast radius.** How many active installations were affected? Was the compromised version distributed through the official WordPress.org repository, or through a third-party marketplace? Were auto-updates involved?
- **Recommend specific actions.** Generic "update your plugins" advice is insufficient for supply chain incidents. Specify whether the plugin should be updated, replaced, or removed entirely—and whether affected sites need a malware scan or password reset.

8. Glossary of WordPress Security Terms

This glossary defines security-related terms as they are used in the WordPress ecosystem. Terms are listed alphabetically. Where a term has both a general and a WordPress-specific meaning, the WordPress usage is emphasized.

2FA / MFA — Two-factor authentication / multi-factor authentication. A security mechanism requiring two or more verification methods (typically a password plus a time-based code from an authenticator app or hardware key) to access an account. In WordPress, 2FA is implemented through plugins or managed hosting features.

Account takeover (ATO) — An attack in which a threat actor gains unauthorized control of a user's account, typically via credential stuffing, phishing, brute-force attack, or stolen session cookies. In WordPress, administrator account takeover is a critical breach scenario. Mitigated by 2FA, strong passwords, and session monitoring. See also: *credential stuffing*, *session hijacking*, *brute-force attack*.

Action-gated reauthentication — A security mechanism that requires a user to re-verify their identity (usually via password and 2FA) specifically before performing a sensitive or destructive action, such as installing a plugin, deleting a theme, or changing user roles. Also known as "sudo mode."

Admin (role) — The highest default user role in a single-site WordPress installation. Administrators can install plugins, modify themes, manage users, and change site settings. On a Multisite network, the equivalent is Super Admin.

AICPA — American Institute of Certified Public Accountants. A U.S.-based professional association that develops auditing and attestation standards, including the Trust Services Criteria used in SOC 2 reporting.

AI-generated phishing — Phishing content produced using large language models (LLMs) or other generative AI tools, typically more convincing and personalized than template-based phishing. IBM's Cost of a Data Breach Report (2025) attributed 37% of AI-driven breaches to this vector. See also: *phishing*, *prompt injection*.

AI-powered tool — A software tool that uses machine learning or AI techniques to perform tasks such as malware scanning, anomaly detection, code review, or vulnerability assessment. When writing about AI-powered tools, describe what they do concretely rather than attributing cognition or judgment to them. See §3.5.

AIDE (Advanced Intrusion Detection Environment) — A server-level file integrity monitoring tool that builds a database of file attributes and checksums at a known-good baseline state, then detects unauthorized changes by periodic comparison. One of the server-side FIM options referenced in the Benchmark alongside OSSEC and Tripwire. See also: *file integrity monitoring*.

Anomaly detection — A security technique that identifies unusual patterns in system behavior, file activity, network traffic, or user actions that deviate from an established

baseline. In WordPress security, anomaly detection flags unauthorized file changes, suspicious login patterns, or abnormal API activity. See also: *file integrity monitoring*, *malware signature*.

Application password — A feature introduced in WordPress 5.6 that generates unique, revocable passwords for REST API and XML-RPC authentication. By design, application passwords provide scoped credentials that do not expose the user's main login password, can be individually revoked if compromised, and are not valid for logging into the WordPress Dashboard. However, they bypass 2FA, do not expire by default, and persist until manually revoked — making them an attack surface that requires careful management in enterprise environments.

Arbitrary file upload — A vulnerability that allows an attacker to upload files of unrestricted types, potentially including executable PHP scripts. Exploitation can lead to remote code execution. WordPress restricts allowed MIME types in the media uploader, but insecure custom upload handlers in plugins and themes are a common vulnerability class. See also: *remote code execution (RCE)*.

Argon2id — A modern password hashing algorithm designed to resist brute-force attacks. In WordPress, bcrypt is the default (since 6.8), and Argon2id can be enabled via the `wp_hash_password_algorithm` filter on servers with the required PHP extensions (sodium or argon2). Argon2id offers stronger resistance to GPU-accelerated brute-force attacks.

Attack surface — The total set of points where an attacker can attempt to enter or extract data from a system. In WordPress, the attack surface includes login forms, all APIs, file upload handlers, plugin and theme code, and the hosting environment. Reducing the attack surface is a core hardening goal.

Audit logging — The systematic, chronological documentation of security-relevant events, actions, and changes within a software application, system, or network. It acts as a "digital security camera," providing a detailed, tamper-evident record (or audit trail) that answers the critical questions of who did what, when, and where.

Auth cookie — The session cookie WordPress sets when a user logs in. It contains the username, an expiration timestamp, and an HMAC signature derived from the authentication keys and salts in `wp-config.php`. This is a signed (not encrypted) token that allows the user to access the Dashboard without re-entering credentials until the cookie expires or the session is terminated.

Authentication — The process of verifying that a user, system, or process is who or what it claims to be. In WordPress, primary authentication occurs via username and password at `wp-login.php`, with optional 2FA. See also: *authorization*, *2FA / MFA*, *passkey / WebAuthn*.

Authentication keys and salts — A set of secret random strings defined in `wp-config.php` (constants such as `AUTH_KEY`, `SECURE_AUTH_KEY`, `LOGGED_IN_KEY`, and

their corresponding salts) used to sign and validate session cookies and tokens. Rotating these constants immediately invalidates all active sessions. Fresh values can be generated at `api.wordpress.org/secret-key/1.1/salt/`. See also: *auth cookie*, *HMAC*.

Authorization — The process of determining what an authenticated user is permitted to do. In WordPress, authorization is managed through the roles and capabilities system. Distinct from *authentication*: a user may be authenticated (identity confirmed) but not authorized (lacking the required capability) to perform a specific action. See also: *role*, *capability*, *principle of least privilege*.

Auto-update — WordPress's built-in mechanism for automatically applying updates. Since version 3.7, minor (security) releases are applied automatically by default. Major version and plugin/theme auto-updates can be enabled separately.

Backdoor — Malicious code inserted into a WordPress installation—typically through a compromised plugin, theme, or dependency—that allows an attacker to regain access to the site even after the original compromise is remediated. Backdoors may be hidden in plugin directories, uploads folders, or modified core files. See also: *supply chain attack*, *malware*, *rogue commit*.

bcrypt — A password hashing function based on the Blowfish cipher. bcrypt has been the default password hashing algorithm in WordPress core since version 6.8 (April 2025), replacing the older phpass-based hashing. WordPress uses SHA-384 pre-hashing to address bcrypt's 72-byte input limit. Application passwords, password reset keys, and other security tokens use the BLAKE2b algorithm via Sodium. bcrypt is mature and widely supported, though Argon2id offers stronger resistance to GPU-based attacks where available.

BLAKE2b — A fast cryptographic hash function used in WordPress (since version 6.8) to hash application passwords, password reset keys, and other security tokens, implemented via the Sodium PHP extension. See also: *Sodium*, *bcrypt*, *SHA-384*.

Breach — An incident in which an unauthorized party gains access to protected systems, accounts, or data. "Data breach" specifically refers to unauthorized disclosure or exfiltration of sensitive information. When writing about breaches, lead with actionable guidance and constructive lessons learned rather than dwelling on the incident. See §3.1.

Brute-force attack — An attack method that attempts to guess login credentials by systematically trying many combinations. In WordPress, this typically targets the `wp-login.php` form. Mitigated by rate limiting, 2FA, and strong password policies.

Build pipeline — An automated sequence of steps (testing, compilation, packaging, deployment) used to build and release software. In WordPress plugin and theme development, build pipelines using npm or Composer can be a supply chain attack vector if a malicious package is introduced at any stage. See also: *supply chain attack*, *dependency confusion*, *rogue commit*.

Capability — A specific permission assigned to a WordPress user role. Examples include `edit_posts`, `manage_options`, and `install_plugins`. Capabilities can be customized with plugins or code to implement the principle of least privilege.

CDN (Content Delivery Network) — A globally distributed network of servers that caches and delivers web content from locations geographically closer to the user. In WordPress security, CDNs provide DDoS mitigation through traffic scrubbing, TLS termination, and WAF capabilities at the network edge. See also: *WAF*, *DDoS*.

Composer — A dependency manager for PHP, widely used in modern WordPress plugin and theme development to manage third-party libraries. If a Composer package is compromised, all projects that depend on it may be affected—a key supply chain risk. See also: *npm*, *dependency confusion*, *SBOM*, *build pipeline*.

Content Security Policy (CSP) — An HTTP response header that controls which resources (scripts, styles, images) a browser is allowed to load on a page. Effective against XSS attacks. Configured at the server or application level.

CORS (Cross-Origin Resource Sharing) — A browser mechanism that controls which external origins may access resources on a web server via JavaScript. In WordPress, CORS is relevant to REST API security: sites serving API responses to specific external origins (e.g., a decoupled front-end on a different domain) should configure CORS headers explicitly rather than using a wildcard (*). See also: *REST API*.

Credential stuffing — An automated attack that uses username/password pairs leaked from other breaches to attempt login on a target site. Effective against users who reuse passwords across services.

Cross-Site Request Forgery (CSRF) — An attack that tricks an authenticated user into performing an unintended action. WordPress mitigates CSRF through nonces—cryptographic tokens tied to a specific user, action, and time window.

Cross-Site Scripting (XSS) — A vulnerability that allows an attacker to inject malicious scripts into web pages viewed by other users. XSS takes three main forms: *stored*, *reflected*, and *DOM-based*. WordPress provides escaping functions (`esc_html()`, `esc_attr()`, `wp_kses()`) to prevent XSS. See also: *stored XSS*, *reflected XSS*, *DOM-based XSS*.

CVE — Common Vulnerabilities and Exposures. A standardized identifier (e.g., `CVE-2024-1234`) assigned to publicly disclosed security vulnerabilities. CVE numbers are issued by authorized numbering authorities.

CVSS — Common Vulnerability Scoring System. A standardized framework for rating the severity of vulnerabilities on a 0–10 scale. Scores classify vulnerabilities as None (0), Low (0.1–3.9), Medium (4.0–6.9), High (7.0–8.9), or Critical (9.0–10.0).

Cybercriminal — A person who uses computers and networks to commit crimes such as unauthorized access, data theft, fraud, or extortion. Preferred over the ambiguous

term “hacker,” which carries positive connotations in many technical communities. See §5.2.

Dashboard — The WordPress administrative interface, accessed via `/wp-admin/`. Prefer “Dashboard” over “backend” or “admin panel” in user-facing writing.

Deepfake — Synthetic media (audio, video, or images) generated or manipulated by AI to impersonate a real person. Deepfakes are increasingly used in social engineering attacks, including impersonation of executives to authorize access or financial transfers. IBM’s Cost of a Data Breach Report (2025) found deepfake-based social engineering in 35% of AI-driven breaches.

Denial of service (DoS) / Distributed denial of service (DDoS) — An attack that overwhelms a server or service with traffic or resource-exhausting requests, rendering it unavailable to legitimate users. A DoS originates from a single source; a DDoS uses many distributed sources, often a botnet. WordPress sites are commonly targeted via XML-RPC amplification or HTTP flood attacks. Mitigated by WAFs, rate limiting, and CDN-layer traffic scrubbing. See also: *rate limiting*, *WAF*, *XML-RPC*.

Dependency confusion — A supply chain attack in which a malicious package with the same name as a private dependency is published to a public registry, causing build tools to install the malicious version. Relevant to WordPress sites that use Composer or npm for dependency management.

DISALLOW_FILE_EDIT — A WordPress constant (`define( 'DISALLOW_FILE_EDIT', true )`) set in `wp-config.php` that disables the built-in Plugin Editor and Theme Editor in the Dashboard. A baseline hardening measure that prevents attackers who gain admin access from editing PHP files directly through the WordPress interface. Does not prevent file modifications through FTP, SSH, or the plugin/theme update mechanism. See also: *DISALLOW_FILE_MODS*, *hardening*.

DISALLOW_FILE_MODS — A WordPress constant (`define( 'DISALLOW_FILE_MODS', true )`) set in `wp-config.php` that disables all file modification capabilities in the Dashboard, including the Plugin/Theme Editors and the ability to install, update, or delete plugins and themes. A strict superset of `DISALLOW_FILE_EDIT`. Requires an external deployment pipeline (e.g., Composer, Git, or CI/CD) to manage updates. See also: *DISALLOW_FILE_EDIT*, *build pipeline*.

Directory traversal — See *path traversal*.

DOM-based XSS — A type of XSS vulnerability in which the attack payload is injected and executed entirely within the browser through manipulation of the Document Object Model (DOM), without the malicious data being included in the server’s HTTP response. Harder to detect with server-side input filtering. See also: *Cross-Site Scripting (XSS)*, *stored XSS*, *reflected XSS*.

EPSS — Exploit Prediction Scoring System. A model that estimates the probability

(0–100%) that a vulnerability will be exploited in the wild within 30 days of scoring. Published by FIRST alongside CVSS, EPSS helps prioritize remediation by real-world exploitability rather than theoretical severity alone.

Fail2Ban — A server-level intrusion prevention tool that monitors log files and bans IP addresses showing malicious patterns (e.g., repeated failed login attempts). Integrates with WordPress through custom jail configurations.

FedRAMP — The Federal Risk and Authorization Management Program. A United States government-wide program that provides a standardized approach to security assessment, authorization, and continuous monitoring for cloud products and services. Relevant for enterprise WordPress deployments in government and highly regulated sectors.

File integrity monitoring — A security practice that detects unauthorized changes to files by comparing their current state (checksums or hashes) against a known-good baseline. In WordPress, `wp core verify-checksums` verifies core files against official hashes, and `wp plugin verify-checksums` verifies plugin checksums where available. Security plugins can extend this to themes, uploads, and custom files.

FORCE_SSL_ADMIN — A WordPress constant (`define( 'FORCE_SSL_ADMIN', true )`) set in `wp-config.php` that forces all Dashboard and login pages to use HTTPS. A baseline hardening measure that ensures authentication cookies are only transmitted over encrypted connections. Note: `FORCE_SSL_LOGIN` is deprecated and should not be used. See also: *HSTS*, *TLS*.

FUD — Fear, uncertainty, and doubt. A rhetorical strategy that exaggerates threats to motivate action (usually purchasing a product). Avoid FUD in security writing; it erodes trust and impairs informed decision-making.

GDPR — General Data Protection Regulation (EU). A European Union regulation governing the processing of personal data of individuals in the EU/EEA. It sets requirements for lawful processing, transparency, data subject rights, breach notification, and controller/processor responsibilities.

GPU-accelerated brute-force attack — A brute-force password cracking technique that exploits the parallel processing power of graphics processing units (GPUs) to test billions of password candidates per second against captured password hashes. WordPress's default bcrypt hashing and optional Argon2id are designed to remain computationally expensive even under GPU acceleration. See also: *brute-force attack*, *bcrypt*, *Argon2id*.

Hardening — The process of reducing a system's attack surface by disabling unnecessary features, restricting permissions, and applying security configurations. In WordPress, hardening includes setting constants in `wp-config.php`, restricting file permissions, and disabling XML-RPC.

HIPAA — Health Insurance Portability and Accountability Act (U.S.). A U.S. law that in-

cludes privacy and security requirements for protecting protected health information (PHI). HIPAA obligations apply to covered entities and business associates and are implemented through administrative, physical, and technical safeguards.

HMAC — Hash-based Message Authentication Code. A cryptographic construct that combines a hash function with a secret key to produce a signature verifying the authenticity and integrity of a message. In WordPress, the auth cookie is HMAC-signed using the authentication keys and salts defined in `wp-config.php`. See also: *auth cookie*, *authentication keys and salts*.

HSTS — HTTP Strict Transport Security. An HTTP response header (`Strict-Transport-Security`) that instructs browsers to only connect to a site over HTTPS for a specified period. Prevents protocol downgrade attacks and cookie hijacking. Should be deployed with care—once a browser receives an HSTS header, it will refuse non-HTTPS connections until the `max-age` expires.

Incident response — The organized process of detecting, containing, eradicating, and recovering from a security incident. In a WordPress context, incident response may include identifying compromised files, revoking credentials, restoring from a clean backup, and notifying affected users. See also: *breach*, *IoC*, *file integrity monitoring*.

Information disclosure — A vulnerability that allows an attacker to access data they are not authorized to view, such as usernames, file paths, configuration values, version numbers, or error messages. WordPress information disclosure vulnerabilities may expose usernames via the REST API, WordPress version information in HTML meta tags, or server configuration details. See also: *user enumeration*, *REST API*.

Infostealer — A category of malware designed to exfiltrate sensitive data from infected devices, including passwords, session cookies, browser data, and cryptocurrency wallets. Infostealers are a rapidly growing threat vector affecting all web platforms, including WordPress. The Verizon DBIR (2025) found that 30% of systems compromised by infostealers were enterprise-licensed devices, and that stolen credentials from infostealers were a significant driver of credential-based breaches.

Insecure AI-generated code — Code produced by LLMs or AI code-completion tools that contains security vulnerabilities because the model generated syntactically valid but insecure patterns—for example, missing input sanitization, improper use of `$wpdb->prepare()`, or hardcoded credentials. AI-generated code should undergo security review before deployment. See §3.5.

IoC (Indicators of Compromise) — Observable evidence that a system has been compromised, such as unexpected file modifications, unfamiliar user accounts, anomalous outbound network traffic, or known malicious file hashes. In WordPress, common IoCs include injected PHP files in plugin directories, unauthorized admin accounts, and modified core files.

KSES — WordPress's HTML sanitization library, used to strip disallowed tags and attributes from content. The name is a recursive acronym: "KSES Strips Evil Scripts." The primary API functions are `wp_kses()`, `wp_kses_post()`, and `wp_kses_data()`. KSES enforces a context-specific allowlist of HTML elements and attributes, preventing stored XSS in user-submitted content. See also: *Cross-Site Scripting (XSS)*, *allowlist*.

Local file inclusion (LFI) / Remote file inclusion (RFI) — File inclusion vulnerabilities in which insecure use of PHP's `include()` or `require()` with user-supplied input allows an attacker to load and execute a local server file (LFI) or a remote attacker-controlled file (RFI). Exploitation can lead to information disclosure or arbitrary code execution. A recurring vulnerability class in WordPress plugins and themes.

Malware — Malicious software designed to disrupt, damage, or gain unauthorized access to a system. In WordPress, malware commonly takes the form of injected PHP backdoors, JavaScript redirects, SEO spam injections, cryptominers, and phishing pages hosted in the uploads directory. See also: *Infostealer*.

Malware signature — A unique byte pattern, hash, or string used to identify a known piece of malicious software. Signature-based detection matches files or code against a database of known malicious patterns. WordPress security scanners use malware signatures to detect injected backdoors, JavaScript redirects, and other threat patterns. See also: *malware*, *file integrity monitoring*, *anomaly detection*.

ModSecurity — An open-source web application firewall (WAF) module for Apache, Nginx, and IIS that inspects and filters HTTP requests based on configurable rule sets. Represents the server-level WAF tier in WordPress deployments. See also: *WAF*, *virtual patching*.

mu-plugin (must-use plugin) — A WordPress plugin placed in the `wp-content/mu-plugins/` directory that loads automatically on every page request and cannot be deactivated through the Dashboard. Must-use plugins are commonly used for security hardening (e.g., disabling XML-RPC, restricting REST API endpoints, enforcing security headers) because they cannot be inadvertently disabled by administrators. They do not receive automatic updates and must be maintained manually or through a deployment pipeline. See also: *plugin*.

Multisite — A WordPress feature that allows multiple sites to be run from a single WordPress installation, sharing the same database and file system. Security considerations differ from single-site installations, particularly around user roles and network-level settings.

NIST SP 800-53 — A NIST publication (*Security and Privacy Controls for Information Systems and Organizations*) providing a comprehensive catalog of security and privacy controls. Individual controls are cited as identifiers such as `IA-2(1)` (Identification and Authentication — Multi-Factor Authentication for Privileged Accounts). Referenced in compliance discussions for enterprise WordPress deployments.

Nonce — In WordPress, a "number used once"—a cryptographic token used to verify that a request originates from a legitimate, authenticated user and is tied to a specific action. Nonces protect against CSRF attacks. Note: despite the name, WordPress nonces are not single-use; they remain valid for a time window (up to 24 hours, in two 12-hour ticks). This is a frequent source of confusion for developers and auditors.

npm — The Node Package Manager; a package registry and command-line tool for JavaScript and Node.js projects. Used in modern WordPress theme and plugin development to manage build-time dependencies. npm dependency chains are a supply chain attack vector. See also: *Composer*, *dependency confusion*, *build pipeline*, *SBOM*.

Object cache — A server-side caching layer that stores frequently queried data in memory to reduce database load. WordPress includes a built-in object cache that persists only for the duration of a single page request. A persistent object cache (backed by Redis or Memcached via a drop-in plugin at `wp-content/object-cache.php`) stores data across requests, significantly improving performance for database-heavy sites. See also: *transient*.

OWASP Top 10 — A regularly updated list of the ten most critical web application security risks, published by the Open Web Application Security Project. The current edition is the OWASP Top 10:2025. Used as a benchmark for evaluating and improving application security.

Passkey / WebAuthn — A passwordless authentication standard based on public-key cryptography. The user's device generates a cryptographic key pair; the private key never leaves the device, and the server stores only the public key. Passkeys resist phishing, credential stuffing, and replay attacks. WordPress support is available through plugins and is expected to reach core in a future release.

Patch / Patching — A software update that fixes a specific bug or vulnerability. In WordPress, patches are delivered through minor version releases (e.g., 6.5.1) and plugin/theme updates. "Virtual patching" refers to WAF rules that block exploitation of a known vulnerability before a code-level fix is applied.

Path traversal — A vulnerability that allows an attacker to access files outside the intended directory by injecting path sequences such as `../` into file path inputs. In WordPress plugins and themes, path traversal can expose sensitive files such as `wp-config.php`. Also known as directory traversal. See also: *directory traversal*.

PCI DSS — Payment Card Industry Data Security Standard. A security standard published by the PCI Security Standards Council that defines requirements for organizations that store, process, or transmit payment card data. PCI DSS compliance applies to a specific cardholder data environment (CDE) and the controls surrounding it.

Phishing — A social engineering attack that uses deceptive communications (usually email) to trick recipients into revealing credentials, installing malware, or taking other

harmful actions. “Spear phishing” targets specific individuals; “whaling” targets executives.

PHP-FPM (FastCGI Process Manager) — The PHP process manager used in LEMP/LAMP stacks that manages PHP worker processes handling WordPress page requests via FastCGI. The recommended PHP execution mode for production WordPress deployments, providing process isolation, per-pool configuration, and better resource management than alternatives such as mod_php.

PHP security directives — PHP `php.ini` directives that control security-relevant behavior. Three directives appear frequently in WordPress hardening contexts: `display_errors` (controls whether PHP error messages are shown to users; must be `Off` in production to prevent information disclosure), `expose_php` (controls the `X-Powered-By` HTTP header that reveals the PHP version; must be `Off` in production), and `open_basedir` (restricts PHP file operations to specified directory trees, preventing path traversal beyond the WordPress installation). All three are Level 1 hardening controls in the Benchmark. See also: *information disclosure*, *path traversal*, *hardening*.

phpass — A portable PHP password hashing framework that was the default password hashing method in WordPress prior to version 6.8. Based on a modified MD5 scheme with stretching, phpass is considered weaker than modern alternatives. WordPress 6.8 (April 2025) replaced phpass with bcrypt as the default. See also: *bcrypt*.

Plugin — A software extension that adds functionality to WordPress. Plugins run with the same privileges as WordPress core, making them a significant component of the site’s security posture. Always one word, lowercase in running text.

Plugin ownership transfer — The change of ownership of a WordPress plugin, typically through sale or abandonment. Plugin ownership transfers are a known supply chain attack vector: malicious buyers have historically acquired plugins to inject backdoors into existing installations via WordPress’s auto-update mechanism. See §7.7, *supply chain attack*.

PoC (Proof of Concept) — In security, a demonstration or snippet of code that proves a vulnerability is exploitable.

Prevention — Security measures designed to stop attacks or breaches before they occur, including patching vulnerabilities, enforcing strong authentication, and applying hardening configurations. Paired with *resilience*—the ability to recover when prevention fails. See §1.

Principle of Least Privilege (PoLP) — A security principle requiring that users and processes be granted only the minimum permissions necessary to perform their functions. In WordPress, this means limiting admin accounts, restricting file modification capabilities, and using custom roles.

Privilege escalation — An attack or vulnerability that allows a user to gain permissions

beyond their assigned level. *Vertical* escalation raises the privilege level (e.g., Subscriber to Administrator); *horizontal* escalation allows access to another user's account at the same privilege level. A frequent vulnerability class in WordPress plugins that mishandle role or capability checks. See also: *role*, *capability*, *principle of least privilege*.

Prompt injection — An attack that targets AI tools by crafting input designed to override or manipulate the tool's instructions, causing it to perform unintended actions or leak sensitive information. Relevant to WordPress deployments that use AI tools to process user-generated or external content. See §3.5.

Ransomware — A category of malware that encrypts a victim's files, databases, or systems and demands payment for the decryption key. WordPress sites may serve as initial access vectors for ransomware deployment against broader organizational infrastructure. Backup integrity, tested recovery procedures, and incident response capabilities are the primary defenses. The Verizon DBIR (2025) found ransomware present in 44% of breaches. See also: *malware*, *incident response*, *resilience*.

Rate limiting — A technique that restricts the number of requests a client can make to a server within a given time window. In WordPress, rate limiting is applied to login attempts (via plugins or server-level tools like Fail2Ban), REST API endpoints, and XML-RPC to mitigate brute-force and denial-of-service attacks.

Reflected XSS — A type of XSS vulnerability in which malicious script is injected through a URL parameter or form input, included in the server's immediate response, and executed in the victim's browser. Requires tricking the victim into clicking a crafted link. Contrast with *stored XSS*, where the payload is persisted in the database. See also: *Cross-Site Scripting (XSS)*, *stored XSS*, *DOM-based XSS*.

Remote code execution (RCE) — A critical vulnerability class that allows an attacker to execute arbitrary code on the target server. RCE can result from insecure deserialization, arbitrary file upload, command injection, or other input handling flaws. Typically rated Critical (CVSS 9.0–10.0). See also: *arbitrary file upload*, *CVSS*.

Resilience — The capacity of a system or organization to withstand, adapt to, and recover from security incidents or failures. In WordPress security writing, resilience refers to backup strategies, incident response procedures, and recovery capabilities that limit damage when prevention fails. See §1.

Responsible disclosure — A practice in which a security researcher reports a vulnerability privately to the affected vendor, allowing time for a patch before public disclosure. The WordPress Security Team follows this practice for core; the WordPress Plugin Security Team handles plugin-specific vulnerability review, forced updates, and plugin closures on WordPress.org.

REST API — WordPress's built-in API for programmatic access to site data. Sensitive endpoints require authentication. The REST API can expose information (e.g., user enumeration via `/wp-json/wp/v2/users`) if not properly restricted.

Rogue commit — An unauthorized or malicious code change introduced into a software project's version control history, typically through a compromised developer account or unauthorized repository access. A supply chain attack vector in WordPress plugins and themes. See §7.7, *supply chain attack*.

Role — A named collection of capabilities in WordPress. Default roles include Subscriber, Contributor, Author, Editor, and Administrator. Custom roles can be created to implement granular access control.

SBOM (Software Bill of Materials) — A formal, machine-readable record of all the components and dependencies in a software package. SBOMs help organizations manage supply chain risk by identifying vulnerable components within themes, plugins, and core libraries.

SHA-384 — A cryptographic hash function in the SHA-2 family producing a 384-bit digest. WordPress uses SHA-384 to pre-hash passwords before passing them to bcrypt, working around bcrypt's 72-byte input limit so that long passwords receive full protection. See also: *bcrypt*, *BLAKE2b*.

SIEM (Security Information and Event Management) — A centralized platform that aggregates, correlates, and alerts on security event data from multiple sources — including WordPress audit logs, web server logs, and WAF events. Commonly used in enterprise WordPress deployments for compliance monitoring and incident detection. See also: *audit logging*, *incident response*.

Shadow AI — The unsanctioned use of AI tools (chatbots, code assistants, content generators) by employees without organizational approval or oversight. Shadow AI risks include leaking sensitive data to third-party AI providers and introducing unreviewed AI-generated code or content. IBM's Cost of a Data Breach Report (2025) found shadow AI incidents added $200,000 to average breach costs ($670,000 for organizations with high shadow AI prevalence) and that 63% of organizations lack AI governance policies.

Session hijacking — An attack in which a threat actor obtains a valid session cookie (e.g., through XSS, network interception, or infostealer malware) and uses it to impersonate the authenticated user. 2FA does not protect against hijacked sessions because the session is already authenticated.

SIM-swapping — A social engineering attack in which a threat actor convinces a mobile carrier to transfer a victim's phone number to an attacker-controlled SIM card, enabling interception of SMS one-time codes and bypassing SMS-based 2FA. The primary reason SMS-based 2FA is not recommended for WordPress administrator accounts. See also: *TOTP*, *Passkey / WebAuthn*.

Severity rating — A classification of a vulnerability's potential impact and exploitability. In WordPress security writing, severity ratings are derived from the CVSS score (Critical, High, Medium, Low) or equivalent classifications used by Patchstack or the WordPress

plugin repository. Always cite the rating framework and version when referencing a score (e.g., "CVSS 3.1: 8.8"). See §7.2, *CVSS*, *EPSS*.

SOC 2 — System and Organization Controls 2. An assurance framework for evaluating an organization's controls related to the AICPA Trust Services Criteria: Security, Availability, Processing Integrity, Confidentiality, and Privacy. SOC 2 results are delivered as an attestation report (commonly Type I or Type II) covering a defined system scope and time period.

Snuffleupagus — An open-source PHP security extension that provides hardening controls — including safe alternatives for `eval()`, enforced type checking, and cookie protection — that PHP's `disable_functions` directive cannot achieve. Referenced as a Level 2 hardening option in the Benchmark for high-security WordPress environments.

Sodium — The PHP `sodium` extension (libsodium), a modern cryptographic library providing authenticated encryption, key derivation, and hashing. WordPress uses Sodium (since version 5.2) for cryptographic operations, including BLAKE2b hashing of application passwords and security tokens, and Argon2id password hashing when enabled. See also: *BLAKE2b*, *Argon2id*.

SQL injection (SQLi) — An attack that inserts malicious SQL code into queries executed by the database. WordPress mitigates SQLi through the `$wpdb->prepare()` method, which parameterizes queries.

SSRF (Server-Side Request Forgery) — A vulnerability that allows an attacker to cause the server to make HTTP requests to unintended destinations, potentially accessing internal services or metadata endpoints. Classified under A01 (Broken Access Control) in the OWASP Top 10:2025; previously a standalone category (A10) in the 2021 edition. In WordPress, SSRF can occur through unvalidated URL inputs in themes, plugins, or the HTTP API. WordPress core mitigates SSRF by filtering outbound HTTP requests to block loopback and private IP addresses and restricting requests to standard ports.

Stored XSS — A type of XSS vulnerability in which malicious script is saved to the server (e.g., in a database comment, post content, or user profile field) and executed whenever a user loads the affected page. More dangerous than reflected XSS because it does not require the victim to click a crafted link. See also: *Cross-Site Scripting (XSS)*, *reflected XSS*, *DOM-based XSS*.

Supply chain attack — An attack that compromises software through its dependencies or distribution channels rather than targeting the software directly. In WordPress, this can occur through compromised plugins, themes, or build tools. See §7.7 for writing guidance.

Super Admin — The highest privileged user role in a WordPress Multisite network. Super Admins can manage all sites in the network, install plugins and themes, create and delete sites, and manage network-wide settings. Distinct from the Admin role, which on a Multisite network is scoped to a single site and has reduced capabilities. When writing

about Multisite security, always specify whether you mean Admin or Super Admin. See also: *Admin (role)*, *Multisite*, *capability*.

Theme — A collection of template files and stylesheets that control a WordPress site's visual presentation. Themes can introduce security vulnerabilities through insecure coding practices, particularly in custom themes.

Threat actor — An individual or group that attempts to exploit vulnerabilities in systems or people for malicious purposes. Preferred over "hacker" in security writing because "hacker" has positive connotations in technical communities.

TOTP — Time-based One-Time Password. An algorithm (defined in RFC 6238) that generates a short-lived numeric code from a shared secret and the current time. TOTP is the most common 2FA method in WordPress plugins (e.g., via authenticator apps like Google Authenticator or Authy). Codes are typically valid for 30 seconds.

TLS (Transport Layer Security) — The cryptographic protocol that encrypts data in transit between a browser and a web server. TLS is the successor to SSL (Secure Sockets Layer); all SSL versions are deprecated and insecure. Current best practice requires TLS 1.2 or later (TLS 1.3 preferred). In writing, use "TLS" when referring to the protocol technically and "HTTPS" when referring to the user-facing URL scheme. Avoid "SSL" unless referring to the deprecated protocol or a product name (e.g., "SSL certificate" remains common usage). See also: *HSTS*, *FORCE_SSL_ADMIN*.

Training-data poisoning — An attack that corrupts an AI model by inserting malicious, mislabeled, or misleading examples into its training data, causing the model to learn incorrect patterns or fail to detect threats. See §3.5.

Transient — A WordPress caching mechanism that stores temporary data in the database (or in the persistent object cache, when available) with an optional expiration time. In WordPress operations, expired transients are a common source of database bloat. WP-CLI provides `wp transient delete --expired` for cleanup. See also: *object cache*.

UFW (Uncomplicated Firewall) — The host-based firewall management tool for Ubuntu/Debian Linux servers, used to restrict inbound network traffic to WordPress production servers (typically allowing only ports 80, 443, and SSH). Provides a simplified interface to `iptables`/`nftables` rules.

User enumeration — A technique used to discover valid usernames on a WordPress site, exploiting the REST API endpoint `/wp-json/wp/v2/users`, differential login error messages, or the `?author=N` URL parameter. Mitigated by restricting the REST API user endpoint, returning generic login errors, and disabling author archives.

Virtual patching — A WAF rule that blocks exploitation of a known vulnerability at the network or application level, providing protection before a code-level patch is available. Services like Patchstack and Cloudflare offer virtual patching for WordPress.

Vulnerability — A weakness in software, configuration, or process that could be exploited to compromise a system's security. In WordPress, vulnerabilities are categorized by type (XSS, SQLi, CSRF, etc.) and severity (CVSS score).

Vulnerability disclosure — The practice of reporting a discovered vulnerability to the affected party (vendor, maintainer, or a coordinating body). Disclosure may be *responsible* (private notification followed by coordinated public release after a patch is available) or *full/immediate* (public release without prior notification). WordPress ecosystem norms strongly favor responsible disclosure. See also: *responsible disclosure*.

WAF — Web Application Firewall. A security layer that filters and monitors HTTP traffic between a web application and the internet. In WordPress, WAFs may operate at the server level (ModSecurity), application level (Wordfence), or network edge (Cloudflare).

Wordfence — A widely used WordPress security plugin and threat intelligence service offering a web application firewall (WAF), malware scanner, login security, and a vulnerability database. In security writing, Wordfence is an example of an application-level WAF. See also: *WAF*, *virtual patching*.

wp-admin — The URL path and directory for the WordPress Dashboard (e.g., `https://example.com/wp-admin/`). Always written in monospace when referring to the path. Some hardening configurations restrict access to this path by IP address. See also: *Dashboard*, *wp-login.php*.

wp-config.php — The primary WordPress configuration file, located in the site's root directory (or one level above). Contains database credentials, authentication keys, and security constants. File permissions should be restricted to the minimum the deployment requires — typically owner-read-only (`400` or `440`) as the preferred steady state, with `600` or `640` used only when deployment automation requires write access. See §3.7 for guidance on writing about context-dependent configurations.

WP-CLI — The official command-line interface for WordPress. WP-CLI allows administrators to manage WordPress installations without a web browser — performing tasks such as updating plugins, managing users, running database operations, and verifying file integrity (`wp core verify-checksums`, `wp plugin verify-checksums`). Always written as "WP-CLI" (hyphenated, all caps). See also: *wp-admin*.

WP-Cron — WordPress's built-in task scheduling system, which triggers scheduled events (such as publishing scheduled posts, checking for updates, and running cleanup tasks) on page load rather than at fixed intervals. Because WP-Cron depends on site traffic, it may fire late on low-traffic sites or cause performance issues on high-traffic sites. The recommended hardening approach is to disable WP-Cron (`define( 'DISABLE_WP_CRON', true )` in `wp-config.php`) and replace it with a system-level cron job. Without the constant, a system cron runs in addition to page-load triggers rather than replacing them. See also: *wp-config.php*.

wp-login.php — The WordPress login form file and URL endpoint (e.g., `https://example.com/wp-login.php`). A frequent target for brute-force attacks. Mitigated by rate limiting, 2FA, CAPTCHA, and IP allowlisting. Always written in monospace. See also: *brute-force attack*, *wp-admin*.

xmlrpc.php — The file implementing the XML-RPC interface in WordPress, located at the site root. A frequent target for brute-force amplification attacks (the `system.multicall` method allows batching multiple login attempts per request). Modern WordPress configurations disable this endpoint at the web server level or via a must-use plugin (`add_filter( 'xmlrpc_enabled', '__return_false' )`). Note: `XMLRPC_REQUEST` is a read-only internal constant that WordPress sets during XML-RPC processing — it cannot be used in `wp-config.php` to disable the feature. Always written in monospace. See also: *XML-RPC*, *XXE*.

XML-RPC — A legacy remote procedure call protocol in WordPress (`xmlrpc.php`). Historically used for remote publishing and pingbacks, it is a common target for brute-force amplification attacks. WordPress core mitigates XXE attacks by disabling custom XML entity loading in the XML-RPC handler. Recommended to disable the endpoint entirely unless specifically required. See also: *XXE*.

XXE (XML eXternal Entity) — An attack against XML parsers that exploits external entity declarations to read local files, perform server-side request forgery, or cause denial of service through entity expansion ("billion laughs"). WordPress core mitigates XXE by disabling the loading of custom XML entities in its XML-RPC handler. Disabling XML-RPC entirely provides defense in depth. See also: *XML-RPC*, *xmlrpc.php*.

Zero-day — A vulnerability that is publicly unknown and unpatched—the vendor has had "zero days" to address it. In formal writing, prefer "previously undisclosed vulnerability" or "publicly undisclosed vulnerability." Hyphenate when used as an adjective: a zero-day vulnerability.

Zero Trust — A security model that requires continuous verification of all users and devices, regardless of network location. No user or system is trusted by default. In WordPress, Zero Trust principles inform practices like requiring 2FA, enforcing session limits, and restricting admin access by IP or device.

9. Operational Appendix: Vulnerability Communication Workflow (Internal)

Plugin/Theme vulnerabilities must always be communicated to customers. To ensure accuracy and consistency, Development provides Customer Success and Marketing with the following details before any public communication:

1. **Vulnerability description and classification** (critical, high, medium, or low severity based on CVSS).

2. **CVE details:** Is there a CVE assigned? Is there a public reference link? Is a proof of concept (PoC) expected?
3. **Patch details:** Patch release version number.
4. **Timeline and scope:** When was the vulnerability introduced? Are all previous versions affected?
5. **Discovery and attribution:** How was the vulnerability found and addressed? Did a security researcher report it to our team or another entity? Is attribution appropriate?
6. **Exploitation history:** Was it exploited in the wild? How can customers check whether they were affected? Are there indicators of compromise (IoC)?
7. **Additional context:** Any technical nuances or environmental factors.

9.1 Vulnerability Communication Workflow (External)

1. **Preparation:** Development provides the information listed above.
2. **Drafting:** Customer Success drafts communications and submits them to leadership for approval.
3. **Coordination:** Development releases the patch and notifies Customer Success and Marketing.
4. **PSA Release Timing:** Determine a Public Service Announcement (PSA) release timeframe that gives users adequate time to update, based on severity and disclosure status.
5. **Execution:** Marketing communicates via internal project management templates and established notification channels.

10. References and Further Reading

Style and Writing Guides

- Bishop Fox Cybersecurity Style Guide — the comprehensive reference for cybersecurity terminology and formatting conventions. This style guide is directly indebted to it.
- Google Developer Documentation Style Guide — general technical writing guidance.
- Microsoft Writing Style Guide — conventions for writing about software interfaces and procedures.

Security Terminology References

- NIST Glossary of Key Information Security Terms
- OWASP Glossary

- SANS Security Glossary

WordPress Security Resources

- Internal WordPress Security White Paper — maintain this reference in your shared document repository (avoid local absolute file paths).
- WordPress Security White Paper (source)
- Hardening WordPress — Advanced Administration Handbook
- Patchstack WordPress Vulnerability Database
- Wordfence Intelligence Vulnerability Database — real-time vulnerability data with proof-of-concept details.
- WPScan Vulnerability Database — now maintained by Automattic as part of the Jetpack ecosystem.

Related Documents

- **WordPress Security Architecture and Hardening Guide** — Enterprise-focused security architecture and hardening guide covering threat landscape, OWASP Top 10 coverage, server hardening, authentication, supply chain, incident response, and AI security.
- **WordPress Security Benchmark** — Prescriptive, auditable hardening controls for the full WordPress stack. Use for compliance verification and configuration audits.
- **WordPress Operations Runbook** — Operational procedures template for WordPress sysadmins and SREs, covering deployment, maintenance, backup, incident response, and disaster recovery.
- **WordPress Security White Paper (WordPress.org, September 2025)** — The official upstream document describing WordPress core security architecture, maintained at wordpress.org/about/security/ (source repository).

License and Attribution

Sources and Acknowledgments: Terminology and formatting conventions are adapted from and indebted to the *Bishop Fox Cybersecurity Style Guide* (2023), used with attribution. AI threat data draws on the Verizon Data Breach Investigations Report (2025) and IBM's Cost of a Data Breach Report (2025).