// HARDENING

A WordPress hardening checklist that actually closes doors

The hardening steps I apply on every threatover engagement. None of them are clever; together they remove most of the easy ways into a WordPress install.

Patrik Grobshäuser · 2026-05-15

> **TL;DR.** Disable file editing, restrict PHP execution in uploads, force 2FA on every admin, rotate AUTH_KEY values, audit and prune plugins, run a real backup (and test restoring). Twenty minutes of work that closes most of the doors attackers use.

After a cleanup, the work that prevents the next one is boring. None of these steps are clever. None of them require a plugin. They collectively close most of the doors I see attackers walking through in 2026.

Where the easy hits come from

Almost every compromise I clean has the same root cause: a plugin or theme with a known vulnerability that wasn't updated, or a credential that was already in a breach list. The expensive part of hardening is not adding exotic protections — it's removing the obvious gaps.

File-level hardening

Disable in-dashboard file editing. Add this to `wp-config.php`:

```
define('DISALLOW_FILE_EDIT', true);
define('DISALLOW_FILE_MODS', true);
```

The first one removes the Theme/Plugin file editor from the admin. The second one prevents installs and updates from the dashboard. The first is always safe. The second is restrictive — only enable it if you have a clear update process (Composer, deploy pipeline) for the site.

Block PHP execution in uploads. The uploads directory should serve images, PDFs, and videos, never PHP. For Apache:

```
# wp-content/uploads/.htaccess
<FilesMatch "\.(php|phtml|php5|pht|phar)$">
    Require all denied
</FilesMatch>
```

For Nginx, deny `.php*` extensions inside the `location ^~ /wp-content/uploads/` block.

This single line stops a large class of "drop a web shell" attacks dead in their tracks. Many attackers don't have a second-stage that doesn't rely on PHP execution from uploads.

Tighten file permissions. Directories `755`, files `644`. `wp-config.php` should be `640` or `600`. The web server user should not own the entire WordPress tree — it should be able to read everything and write only to `wp-content/uploads/` and (for some configurations) `wp-content/cache/`.

wp-config secrets

Rotate the authentication keys. The eight `AUTH_KEY` / `SECURE_AUTH_KEY` / `LOGGED_IN_KEY` / `NONCE_KEY` constants (and their `*_SALT` siblings) sign session cookies. If any of them are old enough that you don't know where they were generated, regenerate them now:

```
https://api.wordpress.org/secret-key/1.1/salt/
```

Paste the output into `wp-config.php`. Every existing login session will be invalidated, which is the point.

While you're there: set `WP_DEBUG` to `false` in production, and make sure `WP_DEBUG_LOG` either isn't set or points to a file that is **not** under the document root.

Login surface

Two-factor on every admin. Pick a plugin that supports TOTP or WebAuthn and turn it on for every administrator. Optional 2FA is the same as no 2FA — the one admin that didn't enable it is the one that gets phished.

Username discipline. Don't use `admin` as a username. If you have an existing user named `admin`, create a new admin with a different name, give it the same capabilities, log out, log back in as the new user, and delete `admin` (assigning their content to your new user).

Limit login attempts. WordPress core does not rate-limit `wp-login.php` and `xmlrpc.php` is even worse. Either block at the edge (a CDN or WAF) or run a plugin that does it. Without this, you are paying for the CPU cycles of every credential-stuffing botnet on the internet.

Disable XML-RPC unless something needs it. Almost nothing on a modern WordPress install needs `xmlrpc.php`. It still ships enabled. Block it at the web server unless you have a specific tool that uses it (the Jetpack/WordPress mobile app are the common cases).

```
location = /xmlrpc.php {
    deny all;
}
```

Plugin discipline

The single best hardening move is fewer plugins. Each installed plugin is a chunk of third-party code that you have not read. The probability of any one of them having an unpatched vulnerability in a five-year window is high.

- Audit your `Active plugins` list once a quarter. Remove anything you don't actively use. "Deactivated" is not "removed" — deactivated plugin files are still on disk and still parseable by request.
- Subscribe to a vulnerability feed. The free options are good enough — Wordfence's Threat Intelligence, Patchstack's database, the WPVulnerability project. Email yourself when a plugin you use gets an advisory.
- When a plugin you use stops getting updates for 12+ months, replace it. Abandoned plugins are an attacker's favourite path; see the Hunk Companion + WP Query Console chain for what that looks like.

Backups that work

You don't have a backup until you have restored from it. I've lost count of how many engagements began with "we have backups, but...".

- Off-site, automated, daily. Local backups don't survive a wiped server.
- Both files and the database, captured at the same point.
- Test a restore at least quarterly. Stand up a staging environment, restore the latest backup, and confirm the site comes up clean. Document the timing.
- Keep at least one backup that is older than 30 days — long enough to predate any compromise that's been waiting to be noticed.

When this stops being enough

If your business depends on a WordPress site staying up, the next level is continuous monitoring rather than periodic hardening: integrity-checking against a known-good filesystem snapshot, alerting on `wp_users` and `wp_options` changes, and logging at the request layer. My Shield plan covers that for $29/site/month.

But the hardening above costs nothing and removes most of the easy hits. Do it once, run a calendar reminder for the recurring parts, and you've taken the cheapest fights off the board.

Got hit anyway? Open a cleanup at threatover.com/contact. Flat $279, 30-day reinfection guarantee.