

FIELD NOTES FROM A PRESALES SOLUTIONS ARCHITECT

Cloudflare Free Plan Security — The Exact Settings I Run on My Own Site

A field note from my own WordPress site on Cloudflare's free plan — what I switched on, what I left off on purpose, and the curl checks that prove each setting is really working.

Daanial Farrukh · 19 October 2026 · <https://quabin.com/cloudflare-free-plan-security-settings/>

I spend a lot of time in customer rooms asking one question about Cloudflare: "Which settings are actually on?"

Most of the time the answer is a shrug and a logo. "We're behind Cloudflare."

So I owe the same honesty about my own site.

This blog runs WordPress on shared hosting, behind Cloudflare, on the **Free plan**. No Pro, no Business, no add-ons. In July a WPScan report on my own site pushed me to do the full pass I normally do for customers.

I expected the free plan to run out quickly. It didn't. It ran out in exactly two places — and I'll show you both.

Everything below is a real screenshot from my own dashboard, taken today.

The Trap — "We're Behind Cloudflare" Is Not a Setting

That's the trap. Turning the orange cloud on gives you a proxy, a free certificate and DDoS protection at the edge. It does **not** give you a strict TLS chain, a WAF policy, security headers or any rule that knows what WordPress is.

Those are switches. Somebody has to flip them, and somebody has to prove they work.

Being behind Cloudflare is a starting point, not a security posture.

What the Free Plan Actually Gives You

Here's the flow every request takes:

Visitor → Cloudflare edge (TLS, WAF, rate limit, headers) → Origin web server → WordPress

On the free plan, the parts you control at the edge are:

- **SSL/TLS** — the encryption mode between Cloudflare and your origin, minimum TLS version, HSTS and HTTPS redirects.

- **Security rules** — up to **5 custom WAF rules** and **1 rate-limiting rule**, plus Cloudflare's managed free ruleset and DDoS protection, which are on by default.
- **Transform rules** — add security headers to every response and strip the headers that leak your stack.

Five custom rules sounds tiny. It's enough — if every rule earns its slot.

Why the Usual Check Misses It

The usual check is opening the site in a browser and seeing the padlock. The padlock only proves the visitor-to-Cloudflare leg. It says nothing about the leg from Cloudflare to your server, nothing about whether `/xmlrpc.php` is reachable, and nothing about which headers leave your origin.

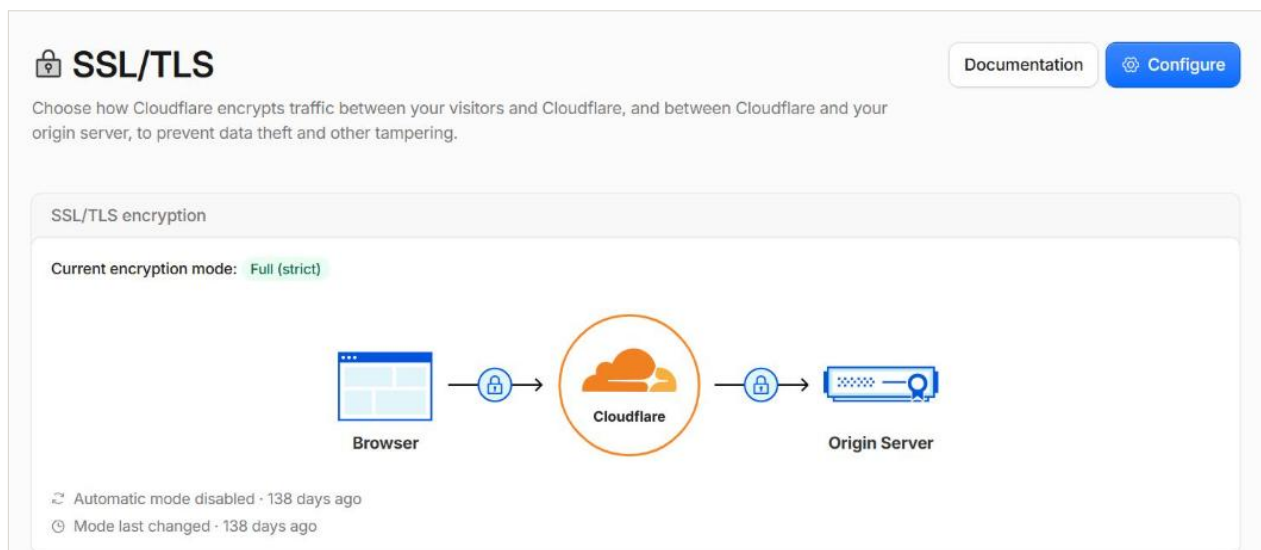
Here's the part that trips people up: Cloudflare will happily serve a green padlock in **Flexible** mode while the traffic to your origin is plain HTTP. The dashboard looks healthy. The chain is not.

So every step below ends with a command, not a screenshot. **The screenshot shows intent. The command shows the truth.**

How I Set It Up — Step by Step

Step 1: SSL/TLS mode — Full (Strict), nothing less

Cloudflare → your domain → SSL/TLS → Overview → Configure → **Full (strict)**.



The SSL/TLS overview on my free-plan zone: current encryption mode Full (strict), browser to Cloudflare to origin, both legs encrypted.

Full (Strict) means Cloudflare encrypts the leg to your origin **and** checks that the origin certificate is valid. If your host has no valid certificate, a free Cloudflare Origin Certificate is enough — SSL/TLS → Origin Server → Create Certificate. My zone has one for the domain and its wildcard, valid until 2041.

Mine was already on Full (Strict). That was the one thing I didn't have to fix.

Plain takeaway: if your mode says Flexible, fix that before you touch anything else on this page.

Step 2: HTTPS everywhere, TLS 1.2 minimum, HSTS without preload

SSL/TLS → Edge Certificates:

- **Always Use HTTPS** — on.
- **Minimum TLS Version** — TLS 1.2. Mine was still on TLS 1.0, the old default. That was a real finding.
- **TLS 1.3** — on.
- **HSTS** — on, max-age 6 months, include subdomains on, no-sniff on, **preload off**.

The screenshot shows the Cloudflare Edge Certificates settings interface. The top section is for HSTS (HTTP Strict Transport Security) settings. It displays: Status: On, Max-Age: 6 months (Recommended), Include subdomains: On, and Preload: Off. A note states 'This setting was last changed 3 months ago'. A blue button labeled 'Change HSTS Settings' is on the right. Below this section are links for 'API' and 'Help'. The bottom section is for 'Minimum TLS Version'. It has a title 'Minimum TLS Version' and a description: 'Only allow HTTPS connections from visitors that support the selected TLS protocol version or newer.' A dropdown menu is set to 'TLS 1.2'. Similar to the top section, there are 'API' and 'Help' links at the bottom.

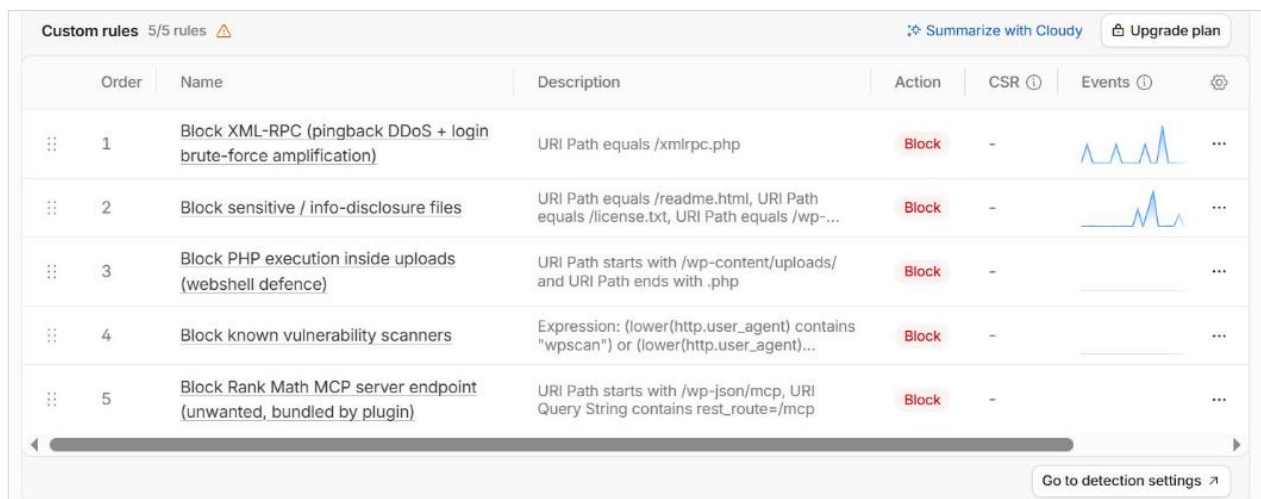
Edge Certificates: HSTS on with a 6-month max-age and subdomains included, preload off, and the minimum TLS version set to 1.2.

Preload is off on purpose. Once a domain is on the browser preload list, getting it off takes months. I want HSTS to stay reversible while the site is still changing hosts and subdomains.

Plain takeaway: change minimum TLS to 1.2 today. Turn on HSTS only after every subdomain works on HTTPS.

Step 3: Five WAF custom rules — every slot earns its place

Security → Security rules → Custom rules → Create rule. The free plan gives you five. I used all five.



The screenshot shows the Cloudflare Custom Rules interface. At the top, it says 'Custom rules 5/5 rules' with a warning icon. There are links for 'Summarize with Cloudy' and 'Upgrade plan'. Below is a table with 5 rules, all set to 'Block'.

Order	Name	Description	Action	CSR	Events
1	Block XML-RPC (pingback DDoS + login brute-force amplification)	URI Path equals /xmlrpc.php	Block	-	
2	Block sensitive / info-disclosure files	URI Path equals /readme.html, URI Path equals /license.txt, URI Path equals /wp-...	Block	-	
3	Block PHP execution inside uploads (webshell defence)	URI Path starts with /wp-content/uploads/ and URI Path ends with .php	Block	-	
4	Block known vulnerability scanners	Expression: (lower(http.user_agent) contains "wpscan") or (lower(http.user_agent)...	Block	-	
5	Block Rank Math MCP server endpoint (unwanted, bundled by plugin)	URI Path starts with /wp-json/mcp, URI Query String contains rest_route=/mcp	Block	-	

At the bottom right of the table, there is a button that says 'Go to detection settings'.

Security rules on the free plan: five of five custom rules in use, all set to Block.

- 1 **Block XML-RPC.** Path equals /xmlrpc.php. It is the classic pingback-DDoS and login-amplification door, and nothing on my site needs it.
- 2 **Block info-disclosure files.** readme.html, license.txt, wp-config.php, the install and setup scripts, anything with /.env or /.git/, and files ending in .sql, .bak or .log.
- 3 **Block PHP inside uploads.** Path starts with /wp-content/uploads/ and ends with .php. Uploads should only ever hold images and PDFs. A PHP file there is a webshell.
- 4 **Block known scanners by user agent.** wpscan, sqlmap, nikto, masscan, nuclei, wfuzz, dirbuster — matched on the lower-cased user agent.
- 5 **Block an endpoint I never asked for.** More on this one below — it was the most surprising finding of the whole pass.

The expression for rule 3, so you can copy it:

```
(starts_with(http.request.uri.path, "/wp-content/uploads/") and ends_with(http.request.uri.path, ".php"))
```

Rule 4 is the weakest of the five. Any serious attacker changes the user agent. It still removes the lazy automated noise from my logs for free, so it keeps its slot.

Step 4: The fifth rule — the MCP server a plugin shipped without asking

During the same pass I found a REST route on my own site at /wp-json/mcp/mcp-adapter-default-server. I had not installed it.

It turned out my SEO plugin, Rank Math, bundles the official WordPress MCP adapter library and registers it on update. It needs authentication — anonymous requests get a 401 — and it pairs with the Abilities API that now sits in WordPress core. But it can execute abilities, and it accepts **Application Passwords, which bypass 2FA**. There was no toggle in the plugin to turn it off.

I want to be fair here: nothing was compromised. My Application Passwords list was empty at the time, so nothing had ever been authorised to use it. It was simply surface I didn't need.

So I closed it twice:

- **At Cloudflare:** rule 5 blocks paths starting with /wp-json/mcp and any query string containing rest_route=/mcp.
- **At the origin:** a small must-use plugin removes the mcp routes through WordPress's rest_endpoints filter, so the namespace is gone from /wp-json/ even if someone reaches the server directly.

Check your /wp-json/ index for namespaces you don't recognise. Plugins add surface on update, quietly.

Step 5: Rate limiting — the free plan's honest limit

Security → Security rules → Rate limiting rules. One rule on the free plan. Mine protects the login form:

POST /wp-login.php — more than 3 requests in 10 seconds from the same IP → Block.

Rate limiting rules1/1 rules⚠️

Upgrade plan

Order	Name	Description	Action	CSR ⓘ	Events ⓘ	
⋮1	Rate limit WordPress login (brute-force protection)	URI Path equals /wp-login.php and Request Method equals POST	Block	-		⋮

The single free-plan rate-limiting rule: POST requests to wp-login.php, action Block.

Here's the honest part. On the free plan the counting period is locked to 10 seconds and the block lasts only 10 seconds. The API rejects anything longer as "not entitled". A patient brute-force script that sends two guesses every ten seconds never trips it.

So I treat this rule as a speed bump, not a lock. **The real login defence lives at the origin:** a security plugin's lockout after failed logins, and TOTP two-factor on the only admin account.

Plain takeaway: on the free plan, rate limiting slows a burst. It does not stop a brute-force. Plan for that.

Step 6: Security headers at the edge — add the good ones, strip the leaky ones

Rules → Transform Rules → Modify Response Header. One rule, applied to every response:

- **Adds** X-Frame-Options, Referrer-Policy, Permissions-Policy and Cross-Origin-Opener-Policy.
- **Removes** x-powered-by, platform and panel — headers my host added that advertised details of the hosting stack to anyone who asked.

Doing this at the edge means the headers are the same on cached pages and uncached pages, and I never have to edit server config on shared hosting.

Step 7: The switches I left off — on purpose

This is the section most hardening guides skip.

Settings [View docs](#)

Settings and tools used to customize and enhance Cloudflare's detections and your domain's security posture.

Q bot fight X Filter by groups Clear filters

1 matching setting

Name	Configurations	Status
Bot fight mode 🔗 Detect and challenges bot traffic on your domain. Bot traffic DDoS attacks	@ 1	☐

Bot Fight Mode is off on this zone, deliberately.

- **Bot Fight Mode** — **off**. It challenges SEO crawlers like Ahrefs, Semrush and Moz, and uptime monitors too. This site exists to be found, so I chose targeted WAF rules over a blanket bot challenge.
- **AI crawlers** — **allowed**. Being cited in AI answers is traffic I want. AI Labyrinth stays off for the same reason.
- **HSTS preload** — **off**, as explained in Step 2.

Security level stays at Medium and Browser Integrity Check stays on. Those cost nothing in reach.

Maximum security is not every switch on. It is every switch decided.

Step 8: The origin — where the free plan stops

Cloudflare only protects traffic that goes through Cloudflare. If someone finds your origin IP, they can skip the edge completely.

Two honest notes from the field:

Authenticated Origin Pulls is off on my zone. It makes Cloudflare present a client certificate to your origin, so the origin can refuse anything that isn't Cloudflare. The switch is free. The work is on the origin web server, which has to be configured to demand that certificate — and on shared hosting that is not my server to configure. So I compensate at the origin: the security plugin's firewall runs in its extended mode, XML-RPC is disabled there too as a backup to the edge rule, and unused themes are deleted.

Never lock admin ports to Cloudflare's IP ranges. On a customer's cPanel server, I once locked myself out of WHM. The firewall allowed WHM only from my admin IP. I opened it through a hostname that was proxied by Cloudflare — and Cloudflare proxies the cPanel and WHM ports. The server saw a Cloudflare IP, not mine, and dropped the connection. SSH on port 22 still worked, which was the clue: Cloudflare's normal proxy doesn't carry it.

The fix was not to allow Cloudflare's ranges. That would open WHM to anyone who routes through Cloudflare. The fix was to reach WHM on the origin IP directly, and keep the server's own hostname grey-cloud (DNS only). Public sites stay orange. **Only the admin path skips Cloudflare.**

Verify It — Where Most Tutorials Stop

This step is where most tutorials stop and where mine starts. Replace example.com with your domain. Every command is read-only.

Blocked paths should return 403:

```
curl -s -o /dev/null -w "%{http_code}\n" https://example.com/xmlrpc.php

curl -s -o /dev/null -w "%{http_code}\n" https://example.com/wp-content/uploads/test.php

curl -s -o /dev/null -w "%{http_code}\n" https://example.com/wp-json/mcp/mcp-adapter-default-server
```

On my site all three return 403, and so do /readme.html and /?rest_route=/mcp. The home page returns 200. If a blocked path returns 200 or 404, the rule is not matching — check the expression, then wait 30 seconds and test again with a cache-busting query string like ?x=1, because rule changes take a moment to reach every edge.

A scanner user agent should be refused:

```
curl -s -o /dev/null -w "%{http_code}\n" -A "WPScan v3.8" https://example.com/
```

Expected: 403. The same URL with a normal browser user agent returns 200.

Old TLS should fail, current TLS should pass:

```
curl -s -o /dev/null -w "%{http_code}\n" --tlsv1.0 --tls-max 1.0 https://example.com/
```

Expected: no page at all — a handshake error, and a status of 000. Run it again with --tlsv1.2 --tls-max 1.2 and you should get 200. This error is not a bug. It is the verification.

Headers should be added and leaks removed:

```
curl -s -D - -o /dev/null https://example.com/
```

Look for Strict-Transport-Security: max-age=15552000; includeSubDomains, X-Frame-Options, Referrer-Policy, Permissions-Policy and Cross-Origin-Opener-Policy. And look for what is **not** there: no x-powered-by, no panel, no platform. Use a real GET like this one, not curl -I — a HEAD request can take a different path through your cache rules and fool you.

On Windows, if curl fails with a certificate revocation error before it ever reaches the site, add --ssl-no-revoke. That is a local Windows check, not a problem with your site.

The Free-Plan Checklist I Now Run

- 1 **Encryption mode is Full (Strict).** SSL/TLS → Overview.
- 2 **Minimum TLS is 1.2 and Always Use HTTPS is on.** The --tls-max 1.0 test fails.
- 3 **HSTS is on, preload is a decision.** Strict-Transport-Security is in the response headers.
- 4 **All five custom rules are used, each with a reason.** Every blocked path returns 403.
- 5 **The rate-limit rule exists, and you know it is only a speed bump.** Login lockout and 2FA live at the origin.
- 6 **Headers are added and leaks are stripped at the edge.** No x-powered-by in the response.
- 7 **Every switch left off is written down with its reason.** Bot Fight Mode, AI crawlers, preload.

- 8 **The admin path does not go through Cloudflare, and Cloudflare's ranges are never on an admin allow-list.**

What I Tell Every Customer Now

The free plan is not the weak link. The undecided switch is.

So in every review I ask one question:

"Can you show me, with a command, that each Cloudflare rule you think you have is actually blocking something?"

If the answer is a list of 403s, they're in good shape. If the answer is "it's in the dashboard," we're not done yet.

I write about the questions I ask in real customer rooms — the ones that change architectures and surface the problems no one else noticed. If you found this useful, drop your email below and I'll send the next one when it's ready.

FAQs

Is the Cloudflare free plan enough to secure a WordPress site?

For a small site, it covers a lot: Full (Strict) TLS, HSTS, five custom WAF rules, one rate-limiting rule, managed rules and DDoS protection. It runs short on rate limiting, which is locked to a 10-second window, and it cannot protect an origin that attackers reach directly. Pair it with a login lockout and 2FA at the origin.

Which Cloudflare SSL mode should I use?

Full (Strict). It encrypts the connection from Cloudflare to your server and checks that the server's certificate is valid. A free Cloudflare Origin Certificate on your server is enough to make it work. Flexible mode leaves that leg unencrypted while still showing visitors a padlock.

Why does the free plan's rate limiting not stop brute-force attacks?

On the free plan the counting period and the block duration are both locked to 10 seconds. A slow script that stays under the threshold is never blocked, and a blocked one is free again after 10 seconds. Use it to slow bursts, and rely on an origin-side lockout and two-factor login for real protection.

Should I turn on Bot Fight Mode?

Only if being crawled is not important to you. Bot Fight Mode also challenges SEO crawlers and uptime monitors. If your site depends on search traffic, targeted WAF rules for the paths and tools you actually want to block are the better trade.

How do I check that my Cloudflare WAF rules are working?

Request a path the rule should block with curl and read the status code. A working block rule returns 403. If you see 200 or 404, the expression is not matching; wait about 30 seconds after any change and add a cache-busting query string before you test again.

Should I whitelist Cloudflare IP ranges for cPanel or WHM access?

No. That opens your admin ports to anyone who routes through Cloudflare. Reach admin panels on the origin IP directly, keep the server's own hostname DNS-only (grey cloud), and allow only your admin IP in the firewall.

Read online: <https://quabin.com/cloudflare-free-plan-security-settings/>