

shop.example.com

Sample report · a made-up online shop; all data is illustrative, not a real site

Date: 2026-10-28

Checked against databases on 28 Oct 2026: NVD, OSV, WPVulnerability + actively exploited list (28 Oct 2026)

32/100

 Security score (100 = fewer observed risks)

Possible risks by public signs. Prioritize the findings below with your developer or hosting provider. Checks can be incomplete; this report offers no security guarantees.

Risk summary

 5	Critical	 5 of them are under active attack right now
 3	High	 1 of them is under active attack right now
 8	Medium	
 5	Low and informational	

Click a row to jump to those findings.

Fix these first

⚡ — Actively exploited

1. **Critical** ⚡ PHP 8.4.26 is vulnerable to PHP2Shell — server takeover without a password
2. **Critical** ⚡ The .env file with database, mail and payment passwords is public
3. **Critical** ⚡ The .git folder with the site source code is public
4. **Critical** ⚡ Database or payment secret key (Supabase service role) is visible in site code
5. **Critical** ⚡ AI service secret key (OpenAI) is visible in site code

Critical ⚡ **Actively exploited**

1. PHP 8.4.26 is vulnerable to PHP2Shell — server takeover without a password

Why it is dangerous right now: Since late October 2026 bots have been mass-exploiting PHP2Shell: they run their own code on the server without a password and steal the customer database. CISA lists it as actively exploited.

What we found

Software	PHP
Component	php
Version	8.4.26
Fixed in version	8.4.27
Vulnerabilities	CVE-2026-109618, CVE-2026-109619
Signal	X-Powered-By: PHP/8.4.26
Page	https://shop.example.com/
Note	Version taken from the X-Powered-By header; distribution builds may carry backported fixes
Affected pages	1
Confidence	likely (please confirm)

Steps to fix

1. Update PHP to the fixed release of your branch: 8.4.27, 8.3.36, 8.2.35 or 8.5.12 (for you: 8.4.27 or newer). On shared hosting this is the “PHP version” setting; on your own server, update the php package.
2. PHP 8.1 or older gets no fix: move to 8.3/8.4 and enable a WAF (e.g. Cloudflare) with PHP2Shell rules until then.
3. Hide the PHP version: set `expose_php = Off` in `php.ini`.
4. Look for signs of compromise: unknown .php files in upload folders, new admin users, php processes spawning `sh`, `curl` or `wget`.
5. How to check: `curl -sI https://shop.example.com/ | grep -i x-powered-by` must show no PHP version, and `php -v` on the server must show the fixed release.

Estimated effort: 30–90 minutes; confirm with your provider

Sources: www.cisa.gov/news-events/alerts/2026/10/21/critical-vulnerabilities-php-require-immediate-patching · www.bleepingcomputer.com/news/security/php2shell-security-flaws-now-successfully-exploited-in-the-wild/

Critical

⚡ Actively exploited

2. The .env file with database, mail and payment passwords is public

Why it is dangerous right now: Bots probe for exposed .env and .git files around the clock: the EmeraldWhale campaign alone harvested over 15,000 cloud credentials this way, and the scanning continues.

What we found

Path	/.env
Server response	HTTP 200
Confidence	confirmed

Steps to fix

1. Block hidden files right away. nginx: location ~ /\.(!well-known) { deny all; }
2. Apache (.htaccess): <FilesMatch ".^\. "> Require all denied </FilesMatch>
3. Move .env outside the web root so the web server cannot serve it at all.
4. Rotate EVERY password and key in the file: database, mail, payment gateway, API keys. Assume they are already stolen.
5. Search the access logs for GET /.env to see when it was downloaded.
6. How to check: curl -s -o /dev/null -w "%{http_code}" https://shop.example.com/.env must return 403 or 404.

Estimated effort: 30–90 minutes; confirm with your provider

Sources: www.sysdig.com/blog/emeraldwhale · honeylabs.net/blog/credential-scraping-env-and-git

Critical

⚡ Actively exploited

3. The .git folder with the site source code is public

Why it is dangerous right now: Attackers download the whole code base, secrets included, from an exposed .git in minutes — this is how the EmeraldWhale campaign stole over 15,000 cloud credentials.

What we found

Path	/.git/config
Server response	HTTP 200
Confidence	confirmed

Steps to fix

1. Block .git. nginx: location ~ /\.git { deny all; }
2. Apache (.htaccess): RedirectMatch 404 /\.git
3. Deploy without the .git folder (git archive, a CI build or rsync --exclude .git).
4. Rotate any password or key that was ever committed — the commit history leaked too.
5. How to check: curl -s -o /dev/null -w "%{http_code}"
https://shop.example.com/.git/config must return 403 or 404.

Estimated effort: 30–90 minutes; confirm with your provider

Sources: www.sysdig.com/blog/emeraldwhale · honeylabs.net/blog/credential-scraping-env-and-git

► Technical details

Critical

⚡ Actively exploited

4. Database or payment secret key (Supabase service_role) is visible in site code

Why it is dangerous right now: This key bypasses every access rule: any visitor can read and change the whole database — orders, customers, passwords. A similar mistake (CVE-2025-48757) exposed data in 170+ apps built with Lovable.

What we found

Service	Supabase service_role
Key fragment (redacted)	eyJhbGciOi...[219 chars, redacted]
Found in	https://shop.example.com/app/assets/index-8c1d2f.js
Confidence	likely (please confirm)

Steps to fix

1. Rotate the key now: Supabase — Project Settings → API Keys (service_role / secret); Stripe — Developers → API keys. Treat the old key as stolen.
2. Use only the public key (anon / publishable) in the browser; admin operations belong on the server (Edge Functions, an API route).
3. Enable row level security on every table: alter table <table> enable row level security; and write policies for who sees which rows.
4. Review database and payment logs for requests and exports you did not make.
5. How to check: the site JS files must not contain service_role, sb_secret_ or sk_live_.

Estimated effort: 30–90 minutes; confirm with your provider

Sources: mattpalmer.io/posts/2025/05/statement-on-CVE-2025-48757/ · supabase.com/docs/guides/database/postgres/row-level-security · escape.tech/blog/methodology-how-we-discovered-vulnerabilities-apps-built-with-vibe-coding/

► Technical details

Critical

⚡ Actively exploited

5. AI service secret key (OpenAI) is visible in site code

Why it is dangerous right now: Sites and chatbots built with AI tools often ship the key inside JavaScript; bots harvest such keys automatically and drain the balance within hours. Escape found 400+ exposed secrets across 5,600 such apps.

What we found

Service	OpenAI
Key fragment (redacted)	sk-proj-Xa...[164 chars, redacted]
Found in	https://shop.example.com/wp-content/plugins/shop-ai-assistant/assets/chat.js
Confidence	likely (please confirm)

Steps to fix

1. Revoke the key now in the provider console and issue a new one — treat the old one as stolen.
2. Set a spending limit in the provider console.
3. Move AI calls to the server (a server function or API route): keep the new key only in server environment variables, never in the browser.
4. Check the provider usage log for requests and charges you did not make.
5. How to check: after rebuilding, search the site JS files for the key prefix (e.g. sk-) — there must be no match.

Estimated effort: 30–90 minutes; confirm with your provider

Sources: escape.tech/blog/methodology-how-we-discovered-vulnerabilities-apps-built-with-vibe-coding/

► Technical details

High ⚡ **Actively exploited**

6. Ninja Forms plugin 3.15.2: the flaw is already used to hack sites (CVE-2026-94504)

Why it is dangerous right now: In October 2026 this Ninja Forms flaw is being mass-exploited: an attacker submits a form entry carrying malicious code, and when an admin opens that submission in the WordPress dashboard the code runs with admin rights — enough to create hidden admin accounts and plant a backdoor. Fixed in 3.15.4.

What we found

Software	WordPress plugin ninja-forms
Component	ninja-forms
Version	3.15.2
Found in	WPVulnerability
Total vulnerabilities	1
Vulnerabilities	CVE-2026-94504
Fixed in version	3.15.4
Signal	readme.txt Stable tag 3.15.2
Confidence	confirmed
Main vulnerability	CVE-2026-94504

Steps to fix

1. Until you update, do not open new entries in Ninja Forms → Submissions.
2. Update the plugin to 3.15.4 or newer: Dashboard → Plugins → “Update now”. If no update exists, deactivate it.
3. Open Users → Administrators: delete unknown accounts and reset every admin password.
4. Check the plugin list and wp-content/uploads for unknown files; restore from a clean backup if you see signs of compromise.
5. Turn on auto-updates for this plugin and two-factor login for admins.
6. How to check: the Plugins page shows the fixed version or newer, and a new SiteCheck scan no longer lists this finding.

Estimated effort: 30–90 minutes; confirm with your provider

Sources: www.bleepingcomputer.com/news/security/ninja-forms-plugin-flaw-exploited-to-hack-wordpress-sites/

► Technical details

High

7. Site calls an AI service directly from the browser

What we found

API addresses	OpenAI
Found in	https://shop.example.com/wp-content/plugins/shop-ai-assistant/assets/chat.js
Confidence	likely (please confirm)

Steps to fix

1. Route AI requests through your own server: keep the key there, limit what data is sent and filter personal data before it leaves your site.

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

High

8. HTTP does not redirect to HTTPS

What we found

Address after redirect	http://shop.example.com/
Server response	HTTP 200
Affected pages	1
Confidence	confirmed

Steps to fix

1. Why: Visitors who type the address without https stay on an insecure connection.
2. Add a permanent 301 redirect from every HTTP URL to HTTPS (example below).
3. nginx (server { listen 80; }): return 301 https://\$host\$request_uri;
4. Apache: RewriteEngine On / RewriteCond %{HTTPS} off / RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]
5. How to check: curl -sI http://shop.example.com/ | grep -i location

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

9. Missing Content-Security-Policy header

What we found

Page	https://shop.example.com/
Header	content-security-policy
Pages	https://shop.example.com/, https://shop.example.com/catalog/, https://shop.example.com/cart/
Affected pages	3
Confidence	confirmed

Steps to fix

1. Why: CSP lets the browser run scripts only from your own and trusted domains — the main barrier against injected code (XSS, a hijacked analytics or chat script).
2. List the domains the site loads scripts, styles and frames from (analytics, chat, CDN): DevTools → Network.
3. Enable the policy as Content-Security-Policy-Report-Only (example below), watch violations in the browser console for 1–2 weeks and extend the domain list.
4. When there are no violations, rename the header to Content-Security-Policy.
5. nginx (server {}): `add_header Content-Security-Policy-Report-Only "default-src 'self'; script-src 'self' https://<script-domains>; object-src 'none'; base-uri 'self'; frame-ancestors 'self'" always;`
6. Apache (.htaccess): Header always set Content-Security-Policy-Report-Only "default-src 'self'; script-src 'self' https://<script-domains>; object-src 'none'; base-uri 'self'; frame-ancestors 'self'"
7. How to check: `curl -sI https://shop.example.com/ | grep -i content-security-policy`

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

10. Missing Strict-Transport-Security header

What we found

Page	https://shop.example.com/
Header	strict-transport-security
Pages	https://shop.example.com/, https://shop.example.com/catalog/, https://shop.example.com/cart/
Affected pages	3
Confidence	confirmed

Steps to fix

1. Why: HSTS makes browsers always use HTTPS and prevents interception of the first HTTP request on public Wi-Fi.
2. Make sure the site and all subdomains work over HTTPS.
3. Start with max-age=86400 and raise it to 31536000 after a week without issues (example below).
4. nginx (server {}): add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
5. Apache (.htaccess): Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
6. How to check: curl -sI https://shop.example.com/ | grep -i strict-transport-security

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

11. Missing X-Content-Type-Options header

What we found

Page	https://shop.example.com/
Header	x-content-type-options
Pages	https://shop.example.com/, https://shop.example.com/catalog/, https://shop.example.com/cart/
Affected pages	3
Confidence	confirmed

Steps to fix

1. Why: Without nosniff the browser may guess a file type and run an uploaded file as a script.
2. Send X-Content-Type-Options: nosniff on every response.
3. nginx (server {}): add_header X-Content-Type-Options "nosniff" always;
4. Apache (.htaccess): Header always set X-Content-Type-Options "nosniff"
5. How to check: curl -sI https://shop.example.com/ | grep -i x-content-type-options

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

12. No protection against framing (X-Frame-Options / frame-ancestors)

What we found

Page	https://shop.example.com/
Header	x-frame-options or CSP frame-ancestors
Affected pages	1
Confidence	confirmed

Steps to fix

1. Why: Without it the site can be framed by another page and visitors tricked into clicking your buttons (clickjacking).
2. Send X-Frame-Options: SAMEORIGIN or add frame-ancestors 'self' to Content-Security-Policy.
3. nginx (server {}): add_header X-Frame-Options "SAMEORIGIN" always;
4. Apache (.htaccess): Header always set X-Frame-Options "SAMEORIGIN"
5. How to check: curl -sI https://shop.example.com/ | grep -i x-frame-options

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

13. Cookies without httponly, samesite flags: woocommerce_session

What we found

Page	https://shop.example.com/
Missing	httponly, samesite
Cookie	woocommerce_session
Cookies	woocommerce_session
Affected pages	1
Confidence	confirmed

Steps to fix

1. Why: Without these flags a session cookie can travel over an insecure connection or be read by an injected script.
2. Add the missing flags (httponly, samesite) to cookies woocommerce_session.
3. If a third-party service sets the cookie (analytics, chat, CDN), configure the flags there — check the cookie domain in DevTools → Application → Cookies.
4. PHP: `session_set_cookie_params(['secure' => true, 'httponly' => true, 'samesite' => 'Lax']);`
5. nginx ≥ 1.19.3 (proxy): `proxy_cookie_flags ~ secure httponly samesite=lax;`
6. How to check: `curl -sI https://shop.example.com/ | grep -i set-cookie`

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

14. No DMARC record for shop.example.com

What we found

Domain _dmarc.shop.example.com

Confidence confirmed

Steps to fix

1. Why: DMARC tells mail services what to do with spoofed mail from your domain and sends you reports.
2. Publish a record with p=none and a report address (example below).
3. After 2–4 weeks, if reports show only your servers, move to p=quarantine.
4. _dmarc.shop.example.com. TXT "v=DMARC1; p=none; rua=mailto:dmarc@shop.example.com"
5. How to check: dig +short TXT _dmarc.shop.example.com

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

15. No SPF record for shop.example.com

What we found

Domain shop.example.com

Confidence confirmed

Steps to fix

1. Why: Without SPF, mail from your domain is easy to spoof and real mail lands in spam more often.
2. Get your mail provider's include (Google: _spf.google.com, Microsoft: spf.protection.outlook.com).
3. Publish a single TXT record (example below).
4. shop.example.com. TXT "v=spf1 include:_spf.yandex.net ~all"
5. How to check: dig +short TXT shop.example.com

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Medium

16. Known vulnerabilities in jquery 1.12.4: CVE-2020-11022

What we found

Software	jquery
Version	1.12.4
Confidence	confirmed
Main vulnerability	CVE-2020-11022
Severity score (CVSS)	6.1

Steps to fix

1. Why: Vulnerabilities are published for this version — automated scanners target such sites first.
2. Compare the installed jquery 1.12.4 version with the affected range (CVE-2020-11022).
3. Back up and update jquery 1.12.4 to a fixed version.
4. How to check: <https://nvd.nist.gov/vuln/detail/CVE-2020-11022>

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Low

17. Server header reveals software and version: “Apache/2.4.58 (Ubuntu)”

What we found

Header	server
Value	Apache/2.4.58 (Ubuntu)
Confidence	confirmed

Steps to fix

1. Why: An exact server version helps automated scanners pick known vulnerabilities.
2. Hide the version in web server responses (example below) and update the server if it is outdated.
3. nginx (http {}): server_tokens off;
4. Apache: ServerTokens Prod и ServerSignature Off
5. How to check: `curl -sI https://shop.example.com/ | grep -i '^server'`

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Low

18. X-Powered-By header reveals the platform: "PHP/8.4.26"

What we found

Header x-powered-by
Value PHP/8.4.26
Confidence confirmed

Steps to fix

1. Why: Platform name and version make it easier to pick known vulnerabilities.
2. Turn the header off in the platform or hide it in the web server (examples below).
3. PHP (php.ini): `expose_php = Off`
4. nginx: `fastcgi_hide_header X-Powered-By; proxy_hide_header X-Powered-By;`
5. Express: `app.disable('x-powered-by')`
6. How to check: `curl -sI https://shop.example.com/ | grep -i x-powered-by`

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Low

19. No /.well-known/security.txt file

What we found

Server response HTTP 404
Confidence confirmed

Steps to fix

1. Why: It tells researchers where to report a vulnerability, so you hear about problems before attackers exploit them.
2. Create `/.well-known/security.txt` with a contact and expiry date (example below).
3. Contact: `mailto:security@shop.example.com`
4. Expires: `2027-12-31T23:59:59.000Z`
5. How to check: `curl -s https://shop.example.com/.well-known/security.txt`

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Low

20. No CAA record for shop.example.com

What we found

Domain shop.example.com

Confidence confirmed

Steps to fix

1. Why: CAA allows only chosen certificate authorities to issue certificates for the domain.
2. Add a CAA record for the CA that issues your certificate (Let's Encrypt example below).
3. shop.example.com. CAA 0 issue "letsencrypt.org"
4. How to check: dig +short CAA shop.example.com

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

Info

21. CMS public signal

What we found

Page <https://shop.example.com/>

Software WordPress

Affected pages 1

Confidence confirmed

Steps to fix

1. Confirm the CMS and keep its core, themes and plugins updated using the official updater.

Estimated effort: 30–90 minutes; confirm with your provider

► Technical details

NVD

This product uses the NVD API but is not endorsed or certified by the NVD.

Non-invasive check based on public data · неинвазивная проверка по публичным данным. This product uses the NVD API but is not endorsed or certified by the NVD.