Security & Vulnerability Audit — Zcloths Ecommerce

Audit date: February 2025
Scope: Full codebase, Firebase, API routes, payment flow, auth, and configuration.

Summary

Severity	Count
Critical	3
High	5
Medium	8
Low	6

Critical

1. Payment amount mismatch (coupon vs Razorpay order)

Field	Value
Severity	Critical
File	`src/app/checkout/page.tsx` (lines 219–230, 238–241)
Risk	Server creates Razorpay order for **cart subtotal only** (no discount). Client opens Razorpay with **discounted total** (`finalTotal * 100`). User either pays full amount while UI shows discount, or payment/verification can fail.
Exploit	User applies coupon, sees discounted total, completes payment; either overcharged (full amount) or flow breaks.
Fix	1) Extend create-order API to accept `discount` and optional `couponCode`; 2) Server validates coupon server-side and creates Razorpay order with `verifiedTotal - discount` (in paise); 3) Client passes **only** `orderId` from API and uses API-returned `amount` for display; do not send `finalTotal * 100` from client.
Best practice	All payment amounts must be computed and enforced server-side; client must never dictate the amount charged.

2. Sensitive data stored in Firestore (payment signature)

Field	Value
Severity	Critical
File	`src/app/api/razorpay/verify/route.ts` (lines 107–114)

Field	Value
Risk	`razorpay_signature` is stored in `processed_payments`. Signatures are cryptographic material; storing them increases impact of a Firestore compromise and is unnecessary for idempotency.
Exploit	DB leak or misused admin access exposes signatures; combined with other data, could aid replay or analysis.
Fix	Store only `payment_id`, `order_id`, `user_id`, `verified_at`, `status`. Remove `signature` from the document.
Best practice	Do not persist cryptographic material (signatures, raw tokens) longer than needed for verification.

3. Firestore `isAdmin()` can throw and break rules

Field	Value
Severity	Critical
File	`firestore.rules` (lines 12–15)
Risk	`isAdmin()` uses `get(/databases/$(database)/documents/users/$(request.auth.uid)).data.isAdmin`. If the user document does not exist (e.g. new Google sign-in before client `setDoc`), `.data` on null causes rule evaluation to fail, denying the request.
Exploit	New users or race conditions (sign-in before user doc created) get permission denied on any rule that uses `isAdmin()`.
Fix	Use `get(...).data.get('isAdmin', false) == true` or check existence first: `let userDoc = get(...); userDoc != null && userDoc.data.isAdmin == true`.
Best practice	In security rules, always handle missing documents and optional fields so evaluation does not throw.

High

4. Auth middleware logs token length and UID

Field	Value
Severity	High
File	`src/lib/auth-middleware.ts` (lines 35, 40, 43, 51–52)
Risk	`console.log` of token length and UID in every authenticated request. In production this can fill logs and leak user identifiers; token length can aid fingerprinting.
Exploit	Log aggregation or compromised logs expose which users hit which endpoints and token characteristics.

Field	Value
Fix	Remove or guard all auth-related `console.log`/`console.error` with `process.env.NODE_ENV === 'development'`.
Best practice	Do not log auth tokens (or their length), and minimize logging of UIDs in production.

5. CSRF: requests without Origin/Referer are allowed

Field	Value
Severity	High
File	`src/lib/csrf-protection.ts` (lines 79–88)
Risk	When both `Origin` and `Referer` are missing, `validateOrigin()` returns `true`. Direct API calls (cURL, Postman, server-side) are allowed. Malicious site can trigger requests that omit these headers (e.g. redirects, some fetch modes).
Exploit	Attacker crafts a flow that causes the victim's browser to send a request without Origin/Referer; if auth is via cookie or pre-filled token, request may be accepted.
Fix	For state-changing or sensitive endpoints, require Origin or Referer when present; if both missing, deny (or apply stricter checks, e.g. custom header + server-side token). Do not default to allow for missing headers.
Best practice	Treat "no Origin/Referer" as untrusted for sensitive operations; require explicit allow-list or same-origin.

6. Shiprocket API routes lack CSRF and rate limiting

Field	Value
Severity	High
Files	`src/app/api/shiprocket/create-shipment/route.ts`, `rates/route.ts`, `track/route.ts`
Risk	Create-shipment, rates, and track use `requireAuth()` only. No `requireValidOrigin()`, no rate limiting, no Zod validation. Authenticated abuse (flooding, cross-site use) is easier.
Exploit	Attacker with a valid token can spam create-shipment or rates; or a malicious site can trigger requests in the user's context.
Fix	Add `requireValidOrigin(request)` and `withRateLimit(request, highLimiter/mediumLimiter, user.uid)`; validate request body/query with Zod (or equivalent) on create-shipment and rates.
Best practice	All authenticated APIs that change state or call external services should have CSRF and rate limiting and validated input.

7. Razorpay verify stores full payment payload in idempotency doc

Field	Value
Severity	High
File	`src/app/api/razorpay/verify/route.ts` (lines 107–114)
Risk	Entire verification payload is not stored, but the stored doc includes `signature` (see Critical #2). Duplicate attempt response (lines 65–73) returns `original_verification` and references to payment/order IDs.
Exploit	Over-exposure in API response and in stored docs increases impact of any leak.
Fix	Do not store signature; do not return `original_verification` in response; return only `verified`, `already_processed`, `order_id` (if needed for client).
Best practice	Idempotency records should hold minimal identifiers and timestamps; API responses should not expose internal verification timestamps.

8. COD orders bypass server-side cart/amount verification

Field	Value
Severity	High
File	`src/app/checkout/page.tsx` (lines 181–186)
Risk	Comment states “COD orders bypass server-side cart verification.” Order is created from client `orderData` only; no server check of cart contents, prices, or totals.
Exploit	Manipulated client (DevTools, modified app) could submit COD orders with wrong items, quantities, or totals.
Fix	Add a server API (e.g. `POST /api/orders/validate-cod`) that accepts cart item IDs and quantities, validates against Firestore product prices and business rules, returns allowed subtotal/discount/total; create order only after validation and use server-computed totals.
Best practice	Every order path (online and COD) must validate cart and totals on the server before creating the order.

Medium

9. Rate limiter fails open

Field	Value
Severity	Medium
File	`src/lib/rate-limit.ts` (lines 179–184)
Risk	On Redis/Upstash error, `rateLimit()` catches and returns `null` (allow). Under Redis outage or misconfiguration, all requests are allowed.

Field	Value
Exploit	Attacker can abuse payment or other critical endpoints when Redis is down.
Fix	For critical tiers (e.g. payment), consider failing closed: on error return a 503 or 429 with Retry-After instead of allowing. Document and test behavior.
Best practice	Define fail-open vs fail-closed per tier; protect payment endpoints with fail-closed or fallback in-memory limits.

10. Webhook does not update order status

Field	Value
Severity	Medium
File	`src/app/api/razorpay/webhook/route.ts` (lines 104–155)
Risk	Events like `payment.captured` and `payment.failed` only log; TODOs for updating order status and sending confirmation are not implemented. Order state in Firestore can stay "processing" after payment success/failure.
Exploit	Operational confusion, support burden, and risk of fulfilling unpaid orders or not fulfilling paid ones if client verification fails.
Fix	Implement webhook handlers: on `payment.captured` update order status (e.g. "paid") and trigger confirmation; on `payment.failed` update to "failed" and optionally notify; use idempotency (already in place) and safe Firestore writes.
Best practice	Payment webhooks must be the source of truth for payment status and must drive order and notification state.

11. Test endpoint exposes token prefix in dev

Field	Value
Severity	Medium
File	`src/app/api/test-shiprocket/route.ts` (lines 10–15)
Risk	Returns `tokenPrefix: token.substring(0, 10) + '...'`. If `NODE_ENV` is ever mis-set in production, or route is deployed without env check, this leaks token material.
Exploit	Information disclosure; combined with other flaws could aid token guessing (unlikely but bad practice).
Fix	Remove this route before production, or never return any part of the token; return only `{ success: true }` in dev.
Best practice	Do not expose any segment of secrets in any environment; use separate "health" checks without secrets.

12. Next.js config references missing packages

Field	Value
Severity	Medium
File	`next.config.ts` (lines 35–38)
Risk	`optimizePackageImports` lists `lucide-react`, `date-fns`, `lodash`. These are not in `package.json`. Build may warn or tree-shaking may not apply as intended.
Exploit	N/A (build/config only).
Fix	Either add the packages if used, or remove them from `optimizePackageImports`.
Best practice	Keep config in sync with actual dependencies.

13. Storage rules use hardcoded `(default)` database

Field	Value
Severity	Medium
File	`storage.rules` (line 13)
Risk	`firestore.get(/databases/(default)/documents/...)` hardcodes default DB. If you ever use a non-default Firestore database, storage rules will still read from default and admin check may be wrong.
Exploit	Misconfiguration when using multiple databases.
Fix	Prefer `firestore.get(/databases/$(database)/documents/...)` if Storage rules support it; otherwise document that only default DB is supported.
Best practice	Avoid hardcoding database IDs in rules when variable is available.

14. FCM token merge may overwrite array

Field	Value
Severity	Medium
File	`src/lib/push-notifications.ts` (lines 28–32)
Risk	`setDoc(userRef, { fcmTokens: arrayUnion(currentToken), ... }, { merge: true })`. With Firestore client SDK, `setDoc` + `merge: true` + `arrayUnion` typically appends; behavior should be verified. If it ever replaced the array, previous tokens would be lost.
Exploit	N/A (reliability only).
Fix	Prefer `updateDoc(userRef, { fcmTokens: arrayUnion(currentToken), lastUpdated: serverTimestamp() })` so arrayUnion is clearly an update operation.

Field	Value
Best practice	Use `updateDoc` + `arrayUnion` for appending to arrays; avoid setDoc + merge for array fields unless documented.

15. Login/signup redirect query not used

Field	Value
Severity	Medium
File	`src/app/login/page.tsx` (no use of searchParams); checkout uses `?redirect=/checkout`
Risk	Checkout (and orders) send users to `/login?redirect=...` but login always redirects to `/`. UX: after login user lands on home instead of checkout or order page.
Exploit	N/A (UX only).
Fix	In login (and signup), read `searchParams.get('redirect')`, validate it is same-origin and path-only (e.g. `/checkout`, `/orders/123`), then `router.push(redirect || '/')`.
Best practice	Use redirect parameter for post-login navigation and validate to prevent open redirect.

16. Duplicate type definitions for Order/Review/Coupon

Field	Value
Severity	Medium
Files	`src/types/index.ts`, `src/types/schema.ts`, `src/contexts/order-context.tsx`
Risk	Order, Review, Coupon, Address defined in multiple places with small differences. Can cause bugs when one place is updated and another is not (e.g. paymentId, shipmentId).
Exploit	N/A (maintainability and consistency).
Fix	Centralize in `src/types` (or schema) and re-export; use a single source of truth for API and UI.
Best practice	Single canonical type definition per domain entity.

Low

17. CORS / security headers not explicitly set

Field	Value
Severity	Low
File	Next.js config / middleware (none found)

Field	Value
Risk	No custom CORS or security headers (X-Frame-Options, X-Content-Type-Options, CSP). Next.js and Vercel provide some defaults; explicit policy is missing.
Fix	Add middleware or headers in `next.config.ts` to set X-Frame-Options, X-Content-Type-Options, Referrer-Policy; consider CSP in report-only first.
Best practice	Define security headers explicitly in one place.

18. Coupon validation is client-only

Field	Value
Severity	Low (in addition to Critical #1)
File	`src/services/coupon-service.ts` (client-side Firestore read)
Risk	Coupon validity and discount are computed on the client. Already covered by Critical #1 for payment; for COD, same issue: client can fake coupon.
Fix	Implement server-side coupon validation in create-order and COD order APIs; apply discount only after server validation.
Best practice	All pricing and discount logic must run on the server.

19. Error boundary does not report to monitoring

Field	Value
Severity	Low
File	`src/components/ErrorBoundary.tsx` (lines 37–40)
Risk	Errors are only logged to console; no Sentry or other reporting. Production errors may go unseen.
Fix	In `componentDidCatch`, call Sentry (or similar) if `NEXT_PUBLIC_SENTRY_DSN` is set.
Best practice	Send client errors to a monitoring service with sanitized context.

20. No explicit index for users subcollections

Field	Value
Severity	Low
File	`firestore.indexes.json`

Field	Value
Risk	Queries like `users/{uid}/orders` with `orderBy('createdAt', 'desc')` may rely on single-field indexes. If you add `where` + `orderBy` later, composite index will be required.
Fix	Add composite indexes for any `where` + `orderBy` on `users/{uid}/orders` or addresses when needed; document required indexes.
Best practice	Define indexes for all production query patterns.

21. Shiprocket track response returns `courierName` as ID

Field	Value
Severity	Low
File	`src/app/api/shiprocket/track/route.ts` (line 32)
Risk	Response sets `courierName: shipment?.courier_company_id` (numeric ID), not the courier name. Frontend may show a number instead of name.
Fix	Use a name field from `tracking_data` or map `courier_company_id` to name if available in API response.
Best practice	API response shape should match frontend expectations and naming.

22. Dependency audit not automated

Field	Value
Severity	Low
File	`package.json` (no audit in CI)
Risk	Known vulnerabilities in dependencies may not be caught before deploy.
Fix	Run `npm audit` (or equivalent) in CI and fail on high/critical; schedule periodic updates.
Best practice	Automate dependency checks and treat high/critical as blocking.

Positive findings

- **Razorpay create-order:** Server-side cart validation, Firestore price lookup, and amount in paise.
- **Razorpay verify:** Signature verification, idempotency via `processed_payments`.
- **Razorpay webhook:** Signature verification, idempotency via `webhook_events`, rate limiting.
- **Auth:** Firebase ID token in Authorization header; `requireAuth` used on sensitive routes.
- **Validation:** Zod used for create-order, verify, and webhook payloads.
- **Firestore rules:** Per-user and per-resource rules; products public read, write for admin.
- **.env.example:** Documents required variables and rotation; no secrets committed.

Recommended priority order

1. Fix Critical #1 (payment/coupon amount server-side).
2. Fix Critical #2 (stop storing signature) and #3 (isAdmin() null-safe).
3. Add server-side validation for COD (High #8) and coupon (Critical #1 / Low #18).
4. Harden CSRF (High #5) and add Shiprocket protections (High #6).
5. Reduce auth logging (High #4), then address Medium/Low items.

End of Security & Vulnerability Audit.