ZNotice — Security Posture

Defensive summary of the security controls in place, the risks that are known and accepted, and the incident response checklist.

1. Authentication and sessions

Area	Control
Customer API	API key via `Authorization: Bearer zn_live_...` or `X-API-Key`. Keys are random (`zn_live_` + 32 hex chars) and stored **only as a SHA-256 hash** (`api_keys.key_hash`, unique). The full key is displayed once at creation. Revocation is a soft delete (`revoked_at`) and takes effect immediately.
Dashboard accounts	Email + password, bcrypt (cost 12). Registration requires accepting the Terms/AUP; `accepted_terms_at` is recorded.
Sessions	Express sessions stored in MariaDB. Cookie `znotice.sid`: `httpOnly`, `sameSite=lax`, `secure` in production, 30-day rolling max age. Session is regenerated on login/registration to prevent session fixation.
Session invalidation	On each request the user is reloaded; deleted or suspended users have their session destroyed. A dashboard password change destroys every **other** session (current one kept); a password **reset** destroys **all** sessions.
Account recovery	Self-service password reset over email (Phase 8). See §1.1.
Proxy trust	`trust proxy = 1` behind NGINX so secure cookies and real client IPs work.

1.1 Password reset controls (Phase 8)

- **Hashed, single-use tokens.** A reset link contains 32 random bytes (`crypto.randomBytes(32)`, 64 hex chars); only `sha256(token)` is stored in `password_resets`. A used token can never be replayed.
- **Atomic claim.** Consumption runs a conditional `UPDATE ... SET used_at=NOW() WHERE id=? AND used_at IS NULL AND expires_at>NOW()` **inside** the password-change transaction; `affectedRows` must be exactly `1`. Two concurrent submissions of the same token therefore produce exactly one winner (regression test in `test/integration/passwordReset.test.js`).
- **Short expiry.** Tokens expire 30 minutes after issue. Requesting a new link deletes any prior unused token for that user.
- **No enumeration.** `POST /forgot-password` always redirects to the same `?sent=1` page and never reveals whether the address exists. `GET/POST /reset-password` return the same `400` invalid-link page for unknown, used or expired tokens.
- **Rate limits.** Forgot: **5 / 15 min / IP**. Reset: **10 / 15 min / IP**.
- **Session invalidation.** A successful reset calls `destroyAllSessions(userId)` (exact `$.userId` JSON match), signing the user out everywhere, and sends a password-changed notification (fire-and-forget).
- **Password policy.** Minimum 8 characters; maximum **72 bytes** (the bcrypt truncation point) on registration, reset and account change.
- **CSRF.** Both forms are protected by the synchronizer token, and the reset path never logs its query string at the NGINX edge (see §4.2).

2. CSRF, headers and rate limiting

- **CSRF:** synchronizer-token pattern. `_csrf` (form body) or `x-csrf-token` header is required on all state-changing requests. Exempt: `/api/v1/*` (API-key auth) and `/webhooks/*` (machine-to-machine, mounted before the session/CSRF middleware), plus the IntouchPay callback `POST /payments/intouch/webhook` (machine endpoint, mounted before session/CSRF — see §3.7). Comparison is timing-safe. Token minting is **lazy** (Phase 8): a session token is created only for authenticated requests or the auth form pages (`/login`, `/register`, `/forgot-password`, `/reset-password`), so public pages no longer create session rows (see §2.1).
- **Security headers (helmet):** CSP locked to `'self'` plus the Tailwind CDN and Google Fonts; `X-Content-Type-Options`, `X-Frame-Options`, `Referrer-Policy` also set at the NGINX layer. `x-powered-by` disabled.
- **Rate limits:**
  - Login/register: 10 attempts / 15 min / IP.
  - Forgot password (`POST /forgot-password`): 5 requests / 15 min / IP.
  - Reset password (`POST /reset-password`): 10 submissions / 15 min / IP.
  - Password change (`POST /dashboard/account/password`): 5 attempts / 15 min per IP + user id (slows current-password brute force).
  - All `/api/v1` requests: 120 / min / IP.
  - Sends per API key (and per dashboard user): `send_rate_per_min` (default 60/min), in-memory fixed window.
  - Payment proof uploads (`POST /dashboard/billing/proof/*`): 30 requests / 15 min, keyed by `user:<id>` else `ip:<ip>` (the limiter skips non-POST).
- **Body limits:** JSON and URL-encoded parsers capped at 1 MB; NGINX `client_max_body_size 50m`; multipart proof uploads capped at **5 MB per file** by multer (and re-checked in the service) before CSRF/auth.
- **Inbound webhook auth:** shared secret compared with `crypto.timingSafeEqual`, accepted as a path token or `x-znotice-webhook-secret` header.
- **Outbound webhook auth:** every customer webhook POST carries `X-ZNotice-Signature: sha256=<HMAC-SHA256(secret, rawBody)>` so the receiver can authenticate ZNotice. Secrets are 32 random bytes (64 hex chars), stored per user, and rotatable from the dashboard.

2.1 Session and cache hardening (Phase 8)

- **Lazy CSRF token.** `attachCsrf` exposes `res.locals.csrfToken` as a getter that only mints a token when `needsCsrf(req)` is true (authenticated user, or one of the auth form pages). Before this, Express's `res.locals` copy read every enumerable getter on each render, which created a session row for every anonymous page view. Public pages are now session-free under `saveUninitialized: false`.
- **Anonymous-session pruning.** The reaper deletes expired sessions and anonymous sessions older than 7 days (`sessionsDeleted` in its summary). A one-time cleanup removed ~1,300 rows created before the lazy-token fix.
- **`Cache-Control: no-store`.** Authenticated HTML responses get `no-store` (`server/index.js`), and **all** `/api/v1` responses get it before auth so 401/403 bodies are also uncacheable (`server/routes/api/v1.js`). Static assets are served before the session layer and are unaffected.
- **Error-log redaction.** The global error handler redacts the `token`, `secret`, `key` and `password` query parameters (case-insensitive) before logging, so a reset token or secret cannot reach the journal.

- **NGINX log suppression.** The vhost sets `access_log off` for the exact `location = /reset-password`, so reset tokens in the query string never reach `access.log`.

3. Admin controls and capacity caps

- Admin routes require an authenticated user with `role = admin` (`requireAdmin`); anonymous users are redirected to login, others get 403.
- Admins can suspend/activate users, adjust credits (negative adjustments that would create a negative balance are rejected), approve/fail payments, manage integration orders, force-disconnect sessions, and edit operational settings.
- The **capacity guard** (`max_concurrent_sessions`, default 15) limits pending+connected WhatsApp sessions to protect host RAM. Admins can raise it only with awareness of memory limits.
- Phase 6 adds three queue caps to the settings whitelist: `bulk_max_recipients` (default 1000), `schedule_max_days_ahead` (default 90) and `scheduled_max_pending` (default 5000).
- Phase 10 adds `sender_name_price_cents` (default 1000) and `sender_name_days` (default 30, clamped 1–365 at settlement) to the settings whitelist; both are admin-only.
- Phase 11 adds `templates_max_per_user` (default 200, range 1–10000) to the settings whitelist; admin-only. It bounds per-user template creation.

3.1 Scheduled & bulk sending controls

- **Atomic per-user pending cap.** Enqueue takes a user-row lock (`SELECT ... FOR UPDATE`) around the pending-count check and the batch/item inserts, so concurrent bulk requests for the same user cannot jointly overrun `scheduled_max_pending` (`429 queue_full`).
- **At-most-once claim.** The scheduler moves an item `pending → sending` with a conditional UPDATE *before* calling the engine. Cancels only target `pending`, so a cancel can never race an in-flight send. A crash mid-send is recovered by the reaper as `failed`/`interrupted_by_restart` rather than retried (see §5.9).
- **No credit reservation.** Enqueue only pre-checks the balance; the actual 1-credit charge happens per message at dispatch and is auto-refunded on engine failure. A job can therefore fail item-by-item later without losing credits (see §5.8).
- **Input validation.** `media_url` is capped at 500 characters (shared with immediate send) so an over-long URL can never surface as a DB "data too long" 500; recipients are normalized and deduped; invalid entries reject the whole request before any row is written.
- **Bounded, tenant-scoped listing.** Queue/batch endpoints clamp `limit` and are scoped to the authenticated user (foreign ids return `404`).

3.2 Usage analytics controls (Phase 7)

- **Tenant-scoped queries.** Every customer aggregate filters on `user_id = ?` (the signed-in user or the API-key owner), so one tenant can never read another's analytics. The customer page is behind `requireAuth` and the API endpoint behind API-key auth.
- **Admin-only platform view.** `/admin/analytics` is behind `requireAdmin` (anonymous → login redirect, non-admin → `403`).
- **Parameterized SQL.** All analytics queries use `pool.query(sql, params)` placeholders; there is no string interpolation of user input.

- **Clamped inputs.** `days` is clamped to `1..90` at the route and again in the service; `topUsers` clamps `limit` to `1..50`. Out-of-range values are clamped, not echoed into SQL, which prevents both `INTERVAL NaN DAY` errors and unbounded result/day-array allocation.
- **Read-only.** The analytics service performs only `SELECT`s — it never writes, and there is no background worker. The migration it relies on (`007_analytics_indexes.sql`) is index-only.

3.3 Dedicated sender-name add-on controls (Phases 9–11)

- **Entitlement gate before the engine call.** `POST /dashboard/whatsapp/sender-name` checks `senderName.isSenderNameActive(userId)` (i.e. `users.sender_name_until > NOW()`) **after** the connected-session check and **before** `wpp.setProfileName()` is ever invoked. An inactive/expired add-on redirects to Billing and never touches the WhatsApp engine. The same helper drives the view, so the disabled form and the server enforce the same rule.
- **Server-side price and period.** The purchase route reads `sender_name_price_cents` and `sender_name_days` from the settings cache; the client cannot supply or influence the amount or the period. The amount is written into the transaction at creation.
- **Admin-only settlement.** Entitlements extend only through `payments.markPaid()`, reached from the `requireAdmin`-protected `POST /admin/transactions/:id/paid`. There is no customer-facing route that grants entitlement.
- **Atomic, idempotent purchase.** Checkout locks the user row (`SELECT ... FOR UPDATE`), re-checks for an existing pending `sender_name` transaction, and inserts in the same transaction — concurrent clicks cannot create multiple pending purchases (TOCTOU closed). Settlement uses the single conditional `pending → paid` UPDATE as the concurrency gate, so concurrent admin approvals cannot double-extend the entitlement.
- **Clamped `days`.** `meta.days` is clamped to `1..365` with an explicit finite check; an unset/unparseable value falls back to the configured `sender_name_days`. This prevents both a zero/negative extension and an unbounded one.
- **Rollback safety.** If the entitlement UPDATE throws after the `pending → paid` transition, the transaction is reverted to `pending` (`paid_at = NULL`) and the error rethrown, so an admin can retry and the payment is not lost.
- **Tenant isolation.** `senderName.isSenderNameActive` is called only with the signed-in `req.user.id`; the billing and pay routes scope transactions to `user_id = ?`.
- **Admin override — deliberate, bounded entitlement bypass (Phase 11).** `POST /admin/sessions/:id/sender-name` is behind `requireAdmin` and the global CSRF middleware. It parses the session id as a safe integer, loads the target session with a parameterized `JOIN`, validates the name with the **same** shared `validateSenderName` helper as the customer route, and requires the session to be `connected` before any engine call. It intentionally does **not** call `isSenderNameActive` — support tooling must be able to correct a tenant's profile name without the tenant owning the add-on. This is the **only** sender-name bypass; the customer paywall is unchanged. Engine failures are logged server-side and surfaced to the admin as a generic message (engine-offline gets a distinct friendly message); no raw engine error, stack or secret reaches the UI.

Residual notes.

- The entitlement check is point-in-time at request time. An admin comp or a manual SQL change to `sender_name_until` takes effect on the next request (no cache), but an in-flight change is not re-checked mid-request.

- Changing the sender name mutates the customer's **WhatsApp account profile** and is subject to WhatsApp's own change-rate limits; ZNotice can only surface a generic error when the engine rejects it.
- The public `/pricing` page reads `sender_name_price_cents` / `sender_name_days` at render time (`server/routes/pages.js`), so it reflects admin changes. The `settings` table remains authoritative for what a customer is actually charged (settlement uses the values stored on the transaction).

3.4 Message template controls (Phase 11)

- **Tenant isolation.** Every template query filters on `user_id = ?` in the SQL `WHERE` clause (list, update, delete, prefill), so a user can never read or modify another tenant's template. A foreign/invalid id simply affects 0 rows and `resolvePrefill` returns `null` — no existence oracle.
- **Strict id parsing.** `parseTemplateId` accepts digits only, requires a safe integer `> 0`, and returns `null` otherwise; both the dashboard router and the service use it, so there is no duplicated/looser parsing path.
- **Atomic per-user cap.** Creation runs in one transaction that first locks the user row (`SELECT ... FOR UPDATE`) and then counts the user's templates with a locking read, so concurrent creates cannot jointly exceed `templates_max_per_user` (default 200). Over the cap throws a typed `limit_reached` error and the insert is rolled back.
- **Bounded listing.** `listForUser` caps the result set at **500** rows (`LIMIT ?`, clamped), so a large template set cannot produce an unbounded render.
- **Input validation.** `validateTemplate` rejects control characters and line breaks in the name and media URL, trims/collapses the name, requires a non-empty body ≤4096 chars and an `http(s)` media URL ≤500 chars, mirroring the send pipeline's limits. Names are unique per user (`UNIQUE(user_id, name)`; duplicate → typed `duplicate_name`).
- **Admin-only cap.** `templates_max_per_user` is part of the admin settings whitelist (range 1–10000) and is never client-supplied.

Phase 11 review findings — all resolved before deploy. No Critical or High issues were identified. **P11-01** (Medium) unbounded template creation → fixed by the atomic per-user cap; **P11-02** (Low) unbounded list → fixed by `LIMIT 500`; **P11-03** strict route id parsing; **P11-04** admin sender-name route safe-integer id; **P11-05** `mediaUrl` control-character rejection.

3.5 Payment proof upload controls (Phase 12)

- **Type decided by magic bytes only.** `sniffMime` (`server/services/proofs.js`) accepts JPEG (`FF D8 FF`), PNG (`89 50 4E 47 0D 0A 1A 0A`), WebP (`RIFF....WEBP`) and PDF (`%PDF-`); the client MIME type and filename are **never** trusted for the type decision.
- **Generated names + regex path guard.** `makeStoredName` produces `<16 hex>.<ext>` from `crypto.randomBytes(8)`; `proofPath` re-validates the name against `^[0-9a-f]{16}\.(jpg|png|webp|pdf)$` before joining it to the uploads root. A user-controlled name can never traverse or select a path.
- **Ownership + admin authorization.** The owner routes join `payment_proofs` → `transactions` and filter `t.user_id = req.user.id`; the admin stream route is behind `requireAdmin`. A foreign or missing proof is an indistinguishable `404` (no existence oracle).
- **CSRF ordering rationale.** Multer runs **before** `verifyCsrf` because the global CSRF check reads `req.body._csrf` and `express.urlencoded` cannot parse a multipart body. Parsing the upload into `memoryStorage` first makes the hidden token visible, so the normal synchronizer-token check still applies; nothing is written to disk before CSRF, ownership, status, size and bytes are validated.

- **Multer limits.** `{ fileSize: 5 MB, files: 1, fields: 1, fieldSize: 1024, parts: 3 }` — bounded so busboy cannot buffer unbounded text fields/parts in memory before CSRF/auth (hardening **H1**).
- **Rate limiting.** 30 POSTs / 15 min under the `/dashboard/billing/proof` mount (the delete POST shares the bucket; GET requests are skipped), keyed by authenticated user id else IP. The limiter runs before multer, so an abusive client cannot make the server buffer files.
- **Per-user quota.** 25 files / 50 MB total across all of a user's transactions, enforced by `getUserProofUsage` on the locked connection before insert (hardening **H2**); plus 5 proofs per transaction and 5 MB each.
- **Transaction row lock + disk I/O.** Ownership/status, the caps and the `INSERT` run under `SELECT ... FOR UPDATE`; the file is written **after** commit, so the row lock is never held across disk I/O (hardening **H4**). A write failure best-effort deletes the committed row and any partial file; no code path unlinks a file after a successful commit.
- **Serving headers.** `Content-Type` comes from the byte-sniffed DB value; `Content-Disposition: inline; filename="proof-<id>.<ext>"` (generated); `Cache-Control: private, no-store`; and `Content-Security-Policy: default-src 'none'; sandbox`. With NGINX's `X-Content-Type-Options: nosniff`, this neutralizes an HTML polyglot served as an inline proof (see §5.16).
- **Upload/delete only while pending.** Both re-assert ownership and `t.status = 'pending'` in SQL; proofs stay readable by owner + admin after settlement for audit.
- **Locked users may still upload.** `/billing` is an allowed prefix under the Phase 13 lock, so an unpaid user can attach proof to the pending payment that unlocks them (hardening **H3**).

3.6 Billing lock for unpaid accounts (Phase 13)

- **Isolation of unpaid accounts.** `requirePayment` (`server/middleware/paymentLock.js`) is mounted at `/dashboard` immediately after `verifyCsrf` and before every dashboard router. A non-admin with no `paid` transaction and no positive credit balance is confined to the payment flow: `/billing`, `/account` and `/integration` (paths relative to the `/dashboard` mount). Every other `/dashboard/*` path gets an error flash and a `302` to the pending transaction's pay page or `/dashboard/billing`.
- **Admin bypass.** `role = 'admin'` passes with no DB read and is never locked. The unlock decision for non-admins is `EXISTS(paid transaction) OR SUM(credit_ledger.delta) > 0` (`server/services/billingAccess.js`), resolved in one query.
- **Anonymous pass-through.** Anonymous requests are not handled here; the downstream `requireAuth` owns the login redirect, so there is a single auth gate and no redirect loop.
- **`/api/v1` intentionally not gated (designed).** The lock is dashboard-only. API-key sends already fail with `402 insufficient_credits` at a zero balance, so gating them would only change the error code while breaking key-based integrations. `/admin/*` is behind `requireAdmin` and is unaffected.
- **Defensive path check.** `isAllowedBillingPath` accepts only strings and returns `false` for anything else, so malformed input can only lock the request down, never unlock it.

3.7 IntouchPay webhook and settlement controls (Phase 14)

The IntouchPay gateway is the first live (non-manual) payment provider. Its callback is fundamentally different from the signed WPPConnect webhook, so the controls differ too.

- **Unsigned callback — no signature verification possible.** IntouchPay does not sign its callbacks, so the app cannot authenticate them. **Mitigation: ground-truth cross-checking.** The handler never

trusts the callback on its own; it always calls `gettransactionstatus` (3 bounded retries) to confirm the gateway actually settled the transaction, and only then settles locally.

- **Pure-ACK response contract.** The webhook always responds `200 {"message":"success","success":true,"request_id":"<uuid>"}` — for a valid, malformed, unknown or mismatched payload alike. This prevents both **validation leakage** (an attacker probing the endpoint learns nothing) and **retry storms** (the provider always sees success, so it never retries a callback that we deliberately did not act on).
- **Secrets stay inside the SHA256 hash.** The `username + account_number + partner_password + timestamp` digest means the account number and partner password are **never transmitted** in any request — only the hash is sent. They live only in `.env` (`INTOUCH_*`).
- **No secrets/tokens in logs.** Webhook payloads and gateway responses are logged without credentials; the `X-Trace-Id` correlation id carries no secret. The global error-log redaction (§2.1) still applies to any `token` / `secret` / `key` / `password` query parameter.
- **Strict amount matching.** Settlement requires the cross-checked status to be settled **and** the returned RWF amount and currency to equal the transaction `meta` exactly (whole RWF, off-by-one blocked), so a manipulated or rounded amount can never settle for the wrong value.
- **Mounted pre-CSR as a machine endpoint.** `POST /payments/intouch/webhook` is mounted **before** the session/CSRF middleware, like `/webhooks/*`, because the provider cannot hold a CSRF token. It requires **no auth** (the callback is unsigned), which is safe precisely because the cross-check is the real trust anchor, not the request itself.
- **Sweep as a recovery control.** `server/services/intouchSweep.js` runs every 5 minutes and cross-checks pending Intouch transactions older than 2 minutes, confirming settled ones. This bounds the window in which a lost/early callback would otherwise leave a settled payment unconfirmed, and it shares the same strict matching + idempotent `pending → paid` gate.
- **No processor refund API — local-ledger reversals only.** IntouchPay exposes no reversal endpoint, so the app never initiates a refund against the gateway. Any refund/adjustment is a local credit-ledger entry (admin adjustment), exactly like `manual` — the same surface is also covered by the existing controls in §3.3/§3.6 and BUSINESS.md §6.
- **Scope.** IntouchPay is credit top-ups only; the sender-name add-on and integration service remain on the manual (admin-approved) flow, so no unsigned-callback settlement path touches entitlements or orders.

3.8 KitovuPay webhook and settlement controls (Phase 15)

The KitovuPay gateway is the second live (non-manual) payment provider. Like IntouchPay, its callback is unsigned; unlike IntouchPay, its outbound requests are HMAC-signed.

- **Unsigned callback — no signature verification possible.** KitovuPay does not sign its callbacks, so the app cannot authenticate them. **Mitigation: ground-truth cross-checking.** The handler never trusts the callback on its own; it always calls `GET /merchant-transaction/payment/:externalId` (3 bounded retries) to confirm the gateway actually settled the transaction, and only then settles locally.
- **Pure-ACK response contract.** The webhook always responds `200 {"message":"success","success":true,"request_id":"<uuid>"}` — for a valid, malformed, unknown or mismatched payload alike. This prevents both **validation leakage** and **retry storms**.
- **HMAC-signed outbound requests.** Every collection POST carries an `x-api-signature` = HMAC-SHA256 over the compact `{method,path,body}` envelope keyed with the API secret; `x-api-timestamp` is Unix milliseconds as a string. The secret only feeds the signature and is **never transmitted**; it lives only in `.env` (`KITOVUPAY_API_SECRET`).

- **No secrets/tokens in logs.** Webhook payloads and gateway responses are logged without credentials (only ids/references).
- **Strict amount matching.** Settlement requires the cross-checked status to be `APPROVED`; a webhook-supplied amount must equal the transaction `meta` exactly (whole RWF, off-by-one blocked), so a manipulated amount can never settle for the wrong value.
- **Unique 6-digit externalId.** The gateway's lookup key is checked for uniqueness in `transactions.provider_ref` before sending, and a `DUPLICATE_EXTERNAL_ID` rejection allocates a fresh ref and retries (bounded) — this also keeps the shared UAT merchant account (with zeron-xs) collision-free.
- **Mounted pre-CSR as a machine endpoint.** `POST /payments/kitovu/webhook` is mounted **before** the session/CSRF middleware and requires **no auth**; the cross-check is the real trust anchor, not the request itself.
- **Sweep as a recovery control.** `server/services/kitovuSweep.js` runs every 5 minutes and cross-checks pending Kitovu transactions older than 2 minutes, confirming settled ones, bounded by the same idempotent `pending → paid` gate.
- **Scope.** KitovuPay is credit top-ups only; the sender-name add-on and integration service remain on the manual flow, so no unsigned-callback settlement path touches entitlements or orders.

4. Host and data handling

- The app runs as `www-data` with systemd hardening (`NoNewPrivileges`, `PrivateTmp`).
- `.env` (DB credentials, session/webhook/engine secrets) is mode `0600` `www-data:www-data` and is never committed.
- Backups include `.env`, so they are stored `0640 root:www-data` with `0750` directories — not world-readable. The backup script never prints credentials and passes DB credentials to `mysqldump` via a temporary `0600` defaults file (never in `argv`).
- SSH is key-based only; UFW allows 22/80/443; Fail2Ban protects SSH.
- MariaDB, the Node app and WPPConnect all bind to loopback; only NGINX is publicly exposed.

4.1 Egress guard (SSRF protection)

Server-side fetches of user-supplied URLs are protected by a shared egress guard (`server/services/netGuard.js`). It:

- allows only `http` / `https` on the default ports (`80` / `443`), rejects URLs with embedded credentials, and rejects `localhost` / `.local` / `.internal` hostnames;
- resolves the hostname and rejects the URL if **any** resolved address is loopback, private, link-local, CGNAT, ULA, multicast, reserved or an IPv6-mapped form of those;
- **pins** the validated IP via a custom socket `lookup` so a DNS-rebinding answer cannot be used between the check and the connect.

The guard is used by **media fetching** (`server/services/mediaFetch.js`, which adds its own content-type/size/redirect limits) and by **outbound customer webhooks** (`server/services/webhookOut.js`, which re-validates the endpoint before every delivery). `media_url` and webhook endpoints therefore share the same public-IP-only rule.

4.2 Host hardening (2026-09-11)

- **UFW:** stale allows for **3001/tcp, 3005/tcp and 4000/tcp** (v4 + v6) were removed, so those app backends are no longer directly reachable from the internet. The GameBet `next-server` on 3001 still serves through NGINX on loopback. Rollback: `ufw allow <port>/tcp`.
- **Sudoers:** a duplicate `kali ALL=(ALL) NOPASSWD:ALL` grant and the empty-group `kali-grant-root` file were removed; `kali` passwordless sudo is retained and `visudo -c` is clean.
- **NGINX:** `access_log off` on the exact `location = /reset-password` keeps live reset tokens out of `access.log` (the pre-existing `/webhooks/` location does the same for the embedded webhook secret).

Full evidence and rollback: OPERATIONS.md §14 and `/root/backups/host-hardening-2026-09-11/CHANGES-AND-ROLLBACK.md`.

5. Known risks (explicitly accepted)

5.1 Unofficial WhatsApp protocol — ban risk

ZNotice uses WPPConnect, an **unofficial** WhatsApp Web automation client. This may violate WhatsApp's Terms of Service and can result in a number being rate-limited, restricted, or permanently banned. This risk is **disclosed to customers in `/terms`** (§3). ZNotice is not affiliated with WhatsApp/Meta. Customers are responsible for consent/opt-in and for complying with the WhatsApp Business Messaging Policy.

5.2 Webhook secret is embedded in the URL path

The engine cannot send custom headers, so the shared secret is part of the webhook URL (`/webhooks/wppconnect/<secret>`). Risks: the secret can appear in engine logs or proxy logs. **Rotation caveat:** rotating `WEBHOOK_SECRET` in `.env` requires an app restart, and existing engine sessions keep calling the **old URL** until they re-run `start-session` (disconnect + reconnect). See OPERATIONS.md §10 "Webhook secret rotation".

5.3 In-memory rate limiter resets on restart

The per-key/per-user send limiter (`services/send.js`) and the WPPConnect bearer-token cache are in-process. A restart clears the send counters, so a burst can briefly exceed the intended per-minute limit. This is acceptable for abuse control on a single node; a distributed limiter (Redis) would be needed for strict enforcement across restarts/replicas.

5.4 Stale pending sessions occupy capacity

A `pending` `wa_sessions` row (user opened the connect flow but never scanned) counts against `max_concurrent_sessions`, potentially blocking new connections. **Mitigation:** the stale-session reaper (`server/services/reaper.js`) runs every 5 minutes and marks `pending` rows untouched for more than 10 minutes as `disconnected` (unless the engine reports `CONNECTED`), and marks `connected` rows `disconnected` when the engine reports `CLOSED`. Admins can still force-disconnect a specific session at **Admin → Sessions**.

5.5 Engine `close-session` 500 bug

`POST /close-session` can return HTTP 500 (`req.client.close is not a function`) **after** the engine has already closed the session. The app mitigates this by probing `status-session` and treating `disconnected` as success. Operators should verify with the status probe before retrying a disconnect; do not assume the session is still live.

5.6 RAM headroom constraints

Each active WhatsApp session (a Chromium/WhatsApp Web context) consumes roughly **300–500 MB**. With ~7.8 GB total RAM and other services on the host, the default cap of 15 sessions is deliberately conservative. Symptoms of pressure: health-check `WARN low available RAM` (<400 MB), app restarts, and engine instability. **Mitigation:** keep the session cap within measured RAM, monitor `free -m`, and force-disconnect idle sessions. Never restart the engine to relieve memory pressure without accepting that all tenants must re-scan.

5.7 Other notes

- `transactions.status` includes `refunded` but no route sets it; refunds are recorded as admin ledger adjustments (see BUSINESS.md §6).
- If `SESSION_SECRET` is not set, the config falls back to a known insecure development value. **Verify `SESSION_SECRET` is set in `.env`** in production.

5.8 No credit reservation at enqueue

Scheduled/bulk jobs do not reserve credits. If the balance is spent before an item dispatches, the item fails with `insufficient_credits` (no credit is lost, since none was charged). This avoids holding funds against jobs that may never run, at the cost of possible partial delivery. **Mitigation:** the enqueue pre-check (`balance ≥ total`) catches the common case; monitor failed items with `last_error='insufficient_credits'`.

5.9 At-most-once dispatch can drop a message

A process crash after the engine accepted a send but before the DB transition leaves the item in `sending`; the reaper marks it `failed` (`interrupted_by_restart`) instead of retrying. This is a deliberate **at-most-once** choice: a rare dropped message is preferred over a duplicate. There is no automatic re-send; operators must confirm with the recipient before re-enqueuing (see OPERATIONS.md §10).

5.10 Scheduler fairness under large backlogs

Each tick selects at most 25 items per user and a global `LIMIT 50`, so with 3+ tenants each holding a large due backlog the third tenant may wait for a later tick. This bounds starvation but does not eliminate latency under sustained overload. **Mitigation:** raise host capacity / `send_rate_per_min`, or spread large bulk jobs over time.

5.11 Anonymous-session growth — mitigated (Phase 8)

Resolved. Previously the CSRF token was attached to every render, so `express-mysql-session` persisted a `sessions` row for anonymous/bot traffic even with `saveUninitialized: false`. Phase 8 makes token minting lazy (§2.1), so public pages no longer create rows, and the reaper now deletes

expired sessions and anonymous sessions older than 7 days. A one-time cleanup removed ~1,300 accumulated rows. Residual risk is minimal; monitor the table size if bot traffic is unusually high.

5.12 L2 — Authenticated responses now send `Cache-Control: no-store` — closed

Closed (Phase 8). Authenticated HTML responses (`server/index.js`) and all `/api/v1` responses including 401/403 (`server/routes/api/v1.js`) now send `Cache-Control: no-store`, so a browser cache / bfcache cannot retain a signed-in page or account-scoped JSON after logout. Static assets are served before the session layer and keep their normal caching.

5.13 Register user enumeration — open (pre-existing)

`POST /register` returns `409` with "An account with that email already exists" for an address that is already registered. This lets an attacker confirm whether an email has a ZNotice account. **Status:** open, pre-existing (not introduced in Phase 8). The forgot-password flow is deliberately non-enumerating; the register response is the remaining asymmetry. A future fix would return the same generic success/verification page regardless of existence (email verification). Until then, treat registered emails as potentially discoverable.

5.14 Forgot-password response timing — open (Low)

`POST /forgot-password` returns an identical status and body whether or not the account exists, but the existing-account path does extra work (lookup + token insert + email) and responds about **15 ms** slower. This is a low-bandwidth timing side channel for user enumeration. **Status:** open; the visual/HTTP response is identical, so it is only observable with careful timing measurement.

5.15 SMTP credentials — configured (closed 2026-09-11)

Production email reuses the existing Hostinger mailbox for `info@consultbofa.com` (`smtp.hostinger.com:465`, implicit TLS); credentials live only in `.env`. Password-reset and password-changed emails are delivered (verified with a live test send). If the variables are ever removed, `send()` returns `email_not_configured` and the reset UI still shows its generic message — see OPERATIONS.md §12.

5.16 Inline PDF proofs — polyglot risk (mitigated)

PDF proofs are served **inline**. A malicious PDF could embed active content, or be a polyglot that is also valid HTML/JS. **Mitigations:** the served `Content-Type` comes from the server-side magic-byte sniff, the response carries `X-Content-Type-Options: nosniff` (NGINX) and `Content-Security-Policy: default-src 'none'; sandbox`, and the `Content-Disposition` filename is generated. A PDF rendered in the browser's sandboxed viewer cannot run page script in the ZNotice origin. **Status:** mitigated; keep the CSP/nosniff headers in place if the serving routes change.

5.17 Open registration + per-account proof quota (open)

Registration is open and there is **no email verification** (see §5.13). The proof quota (25 files / 50 MB per user) is therefore per *account*, not per verified identity, so an attacker can create many accounts to obtain many quotas. **Mitigations in place:** the 30/15-min upload rate limit, magic-byte typing, the storage caps, and the fact that files are only reachable through authenticated routes. **Status:** open, pre-existing; a real fix is email verification / CAPTCHA at registration.

5.18 IPv6 rate-limit granularity (open)

The proof-upload limiter keys anonymous requests on the raw `req.ip`. For IPv6 a client controls a large `/64`, so rotating source addresses within that allocation can evade a per-address bucket. Authenticated uploads are keyed on `user:<id>` and are unaffected. **Status:** open, Low (the route still requires an authenticated session to succeed; the limiter is a secondary abuse control). A stricter key would normalize IPv6 to a `/56` / `/64` prefix.

5.19 Payment-proof files excluded from backups (open)

The automated app archive excludes `uploads/`, so the `payment_proofs` DB rows are backed up but the file bytes are not; a DB-only restore leaves rows pointing at missing files. **Status:** open operational gap — see OPERATIONS.md §15.4 for the retention/backup recommendation. Proof files are the only business data not covered by the backup.

5.20 IntouchPay webhook is unsigned (open, accepted)

The IntouchPay callback carries no signature, so the app cannot authenticate it. This is a limitation of the provider, not a ZNotice omission. **Mitigation:** settlement is gated on a `gettransactionstatus` cross-check (never on the callback alone), and the endpoint returns a pure ACK for every payload, so a forged or garbage callback can at most trigger a status lookup — it can never settle a transaction by itself. **Residual:** the endpoint is publicly reachable and unauthenticated, so it can be probed or spammed; the bounded-retry cross-check and the ACK contract keep the per-request cost low, but an attacker can still cause extra `gettransactionstatus` calls. **Status:** open, accepted — see §3.7 for the full control list.

5.21 KitovuPay webhook is unsigned (open, accepted)

The KitovuPay callback carries no signature, so the app cannot authenticate it. This is a limitation of the provider, not a ZNotice omission. **Mitigation:** settlement is gated on a `GET /merchant-transaction/payment/:externalId` cross-check (never on the callback alone), and the endpoint returns a pure ACK for every payload, so a forged or garbage callback can at most trigger a status lookup — it can never settle a transaction by itself. **Residual:** the endpoint is publicly reachable and unauthenticated, so it can be probed or spammed; the bounded-retry cross-check and the ACK contract keep the per-request cost low, but an attacker can still cause extra status calls. **Status:** open, accepted — see §3.8 for the full control list.

6. Incident response checklist

Use in order; stop when the incident is contained.

1. **Confirm and scope**
   - `journalctl -u znotice --since "1 hour ago"` and `docker logs --tail=200 wppconnect`.
   - Identify affected user(s), API key(s) and session(s) in the database.
2. **Suspend the abusive account**
   - **Admin → Users → Suspend** (or `UPDATE users SET status='suspended' WHERE id=?`).
   - Suspension immediately blocks sends (`403 account_suspended`) and API key authentication.
3. **Rotate credentials / keys**
   - Revoke the affected API keys from **Admin → Users** / user dashboard (soft-delete via `revoked_at`).

- If a secret is suspected leaked, rotate `WEBHOOK_SECRET` (and `WPPCONNECT_SECRET_KEY` if applicable) in `.env`, restart the app, and force reconnect of sessions so the engine picks up the new webhook URL.
- If a dashboard password is compromised, reset it and destroy sessions (restart clears in-memory state; delete the user's rows from the `sessions` table to force logout).

4. **Force-disconnect sessions**
   - **Admin → Sessions → Disconnect** to free capacity slots and stop a session that may be spamming. Verify with the engine status probe.
5. **Preserve evidence**
   - Copy relevant journal lines, engine logs and DB rows before cleanup.
6. **Restore if data integrity is affected**
   - Use the daily backup: restore the database and/or app archive (OPERATIONS.md §5). Prefer a clean restore over patching live data.
7. **Follow up**
   - Re-enable the account only after remediation.
   - Record the incident, root cause, and any setting changes (for example a lower `send_rate_per_min` or `max_concurrent_sessions`).