

SECURITY ASSESSMENT

for

Yimmit Exchange

on

Aug 12, 2026



FIDESIUM

Table of Contents

Executive Summary	3
Methodology & Severity	5
Architecture & Trust Model	6
Risk Issues	7
Remediation Roadmap	14
Vulnerabilities	15
Critical	15
High	20
Medium	23
Low	34
Info	43
Financial-Flow & Engine Findings	46
Provenance, Blast Radius & Integrator Trust	60
Appendix: Positive Observations	62
Appendix: Identity, 2FA & Recovery Surface	66
Appendix: Authorization & Object-Ownership (BOLA)	67
Appendix: Web AppSec Sink Coverage	68
Appendix: Authorization / RBAC Model	69
Appendix: Deposit / Withdraw State Machine	70
Appendix: Vault Channel & ShuftiPro KYC	71
Disclaimer	72

Executive Summary

Report



Abstract

Yimmit engaged Fidesium to conduct a **whitebox security assessment** of the Yimmit Exchange, a centralized cryptocurrency exchange (spot, swaps, staking, P2P, fiat on/off-ramp, referrals) built on a fork of the open-source OpenDAX / Peatio / Barong stack. The engagement includes authorized active testing against the live production deployment at yimmit.com, with **mainnet withdrawal flows in scope**.

Components in scope:

1. **exchange-auth**: Rails 7 authentication / account service (Barong fork, **VERSION 3.2.0**) and the API authorization gateway.
2. **exchange-base**Peatio Fork
3. **exchange-castle**: React/TypeScript administrative dashboard.
4. **frontend-ui**: React 18 / TypeScript user trading frontend.
5. Supporting documentation (architecture, DB schema, admin Swagger, Postman collections, Fireblocks / fiat fund-flow specs).

Key Findings

Secrets in the repository. A live, fine-grained GitHub PAT was committed in the **Gemfile** (**C6** live-confirmed admin/push to 4 prod-dependency repos leading to a supply-chain RCE), granting access to private **yimmitOrg** plugin repositories. Recommended for immediate rotation.

Issue Summary

Critical
6 1 Issues

High
14 1 Issues

Medium
35 10 Issues

Low
26 8 Issues

Info
6 1 Issues

Engagement

Report Version	3
Engagement Type	Whitebox + authorized live testing
Engagement Start	Jun 04, 2026
Engagement End	Jun 28, 2026
Rereview	Jul 17, 2026
Rereview	Aug 04 - 12, 2026

Team

Lead Security Researcher	Abraham Polishchuk @skilurus
--------------------------	------------------------------

Scope

Repositories	5 (exchange-auth, exchange-base, staking, exchange-castle, frontend-ui)
Backend service	Rails 7 / Grape API / MySQL / Redis / Sidekiq (Barong fork)
Live target	yimmit.com (production; mainnet withdrawal flows in scope)
Custody / secrets	Fireblocks (wallets), HashiCorp Vault (secrets)
Identity	Sumsb / ShuftiPro KYC
Out of scope	Volumetric / availability DoS (network floods, sustained L7 floods); third-party providers (Fireblocks, Sumsb, Cloudflare) internals

Review Commits

Repository	Initial review	Secondary review	Tertiary review	Quaternary review
exchange-auth (Barong)	4b0f0c2	—	43746b8	003dc87
exchange-base (Peatio)	bc63e8d	6252ad5	—	6f097c2
staking	c988cab	—	—	a3c2570
exchange-castle (admin)	e146d31	—	1d534d5	6e30ce6
frontend-ui	72e09b4	—	82b30f8	58b126c

Methodology

- Manual source-code review (server-side, whitebox)
- Threat modeling (actors, assets, trust boundaries, entry points)
- Authentication & session-handling analysis
- JWT / key-handling and signature-verification review
- Authorization matrix derivation from source (RBAC / abilities)
- API-key, nonce & HMAC scheme review
- Baseline-fork CVE mapping & upstream differential review
- Secret & configuration audit (working tree + git history)
- Dependency review against pinned lockfiles
- Authorized dynamic testing against the live deployment
- Financial-flow & state-machine analysis (orders, withdrawals, ledger)
- Concurrency / race-condition modeling

Severity Definitions

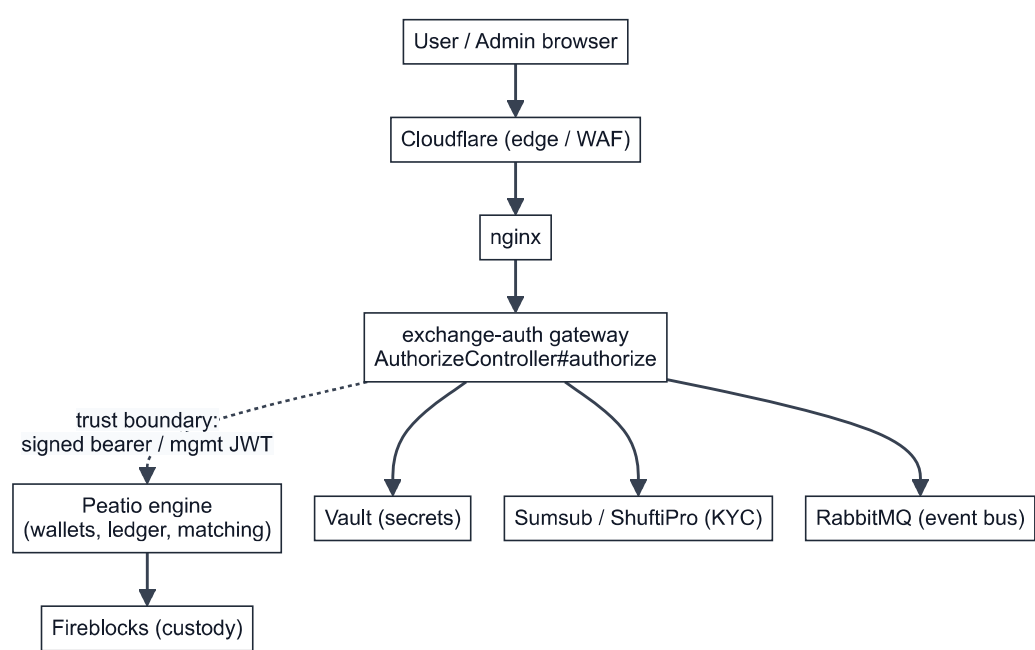
Critical	The issue can cause large economic losses, large-scale data disorder, or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputational damage, or moderate financial impact.
Low	The risk is relatively small, not exploitable on a recurring basis, or low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defense in depth.

Probability reflects exploitation likelihood given current configuration and access. Conditional findings (pending live confirmation) are tagged in the description.

Architecture & Trust Model

Trust Boundaries

The exchange is centralized and custodial. The exchange-auth service is both the identity provider and the API authorization gateway: every request to `/api/v2/auth/*` is authenticated by `AuthorizeController#authorize`, which validates the caller and emits a signed bearer token that upstream services trust. The same keystore key signs the management-API JWTs that instruct the Peatio engine to move funds.



Key Trust Assumptions & Where They Break

Assumption	Reality	Finding
Signing key is secret and prod-only	A valid key-pair is committed to the repo Active <i>[Still open: <code>config/rsa-key</code> remains tracked at HEAD.]</i>	L24
Client IP is trustworthy at the gateway	Derived from leftmost attacker-set <code>X-Forwarded-For</code> Resolved <i>[Re-test 2026-08-04: client IP now resolves only via the trusted-proxy-aware <code>remote_ip</code>, and the <code>CF-Connecting-IP</code> regression it introduced (<code>H15</code>) is now fixed.]</i>	H1
Signed API requests cannot be replayed	Nonce window ~317,000 years Resolved <i>[Updated: nonce lifetime cut to 10s + per-(kid,nonce) Redis SETNX dedup. Replays are now rejected.]</i>	H2
Peatio honors only multi-signed mgmt commands	Threshold is caller-configured Resolved <i>[Updated: <code>config/management_api_v1.yml</code> ships two <code>mandatory_signers</code> per fund scope in-repo, and the middleware fails closed on a missing signer.]</i>	H4
Bearer tokens are signature-verified	Verifier key is nil; relies on a Redis existence gate Resolved <i>[Re-test 2026-08-04: <code>authorize.rb</code> now calls <code>authenticate!</code> → <code>decode_and_verify_token (verify=true, RS256)</code>; the signature is checked and it 401s (M1).]</i>	M1

Risk Issues

☐ Active
 ☐ New
 ☐ Acknowledged
 ☐ Remediated
 ☐ Resolved

Vulnerability	Description	Risk	Probability	Status
C1: Unauthenticated Fireblocks webhook	<code>POST /public/fireblock/webhook</code> credits a member's balance from unauthenticated POST-body params (amount/asset/address).	Critical	High	Resolved
C2: Unauthenticated account takeover via <code>/verify_access_token</code>	<code>GET .../sessions/verify_access_token</code> decodes a caller-supplied JWT without signature verification (<code>decode_token, jwt.rb:40</code> <code>verify=false</code>) and opens a session for the token's email (<code>sessions.rb:782</code>)	Critical	Low	Resolved
C3: Hardcoded Jenkins admin credentials (supply chain)	<code>exchange-base/lib/restart_service/client.rb:9-12</code> commits a Jenkins admin API token + internal IP. Compounds H3	Critical	Medium	Remediated
C4: Committed Event-API RSA private key	<code>exchange-base/config/application.yml:133</code> commits a full RSA private key (<code>EVENT_API_JWT_PRIVATE_KEY</code> default)	Critical	Low	Remediated
C5: Full user-table disclosure	<code>GET /resource/users/records/:platform</code> (<code>resource/users.rb:163</code>) returns the entire <code>users</code> table incl. <code>password_digest</code> , the live second-factor <code>code/verification_code</code> , and full PII/KYC to any authenticated member.	Critical	High	Resolved
C6: Live committed admin GitHub PAT	A fine-grained PAT committed in <code>exchange-auth/Gemfile</code> (+ git history) is live with <code>admin/push</code> to 4 private prod-dependency repos	Critical	High	Active
H1: <code>X-Forwarded-For</code> trust at gateway	Leftmost raw XFF used for IP allowlists, geo/blacklist, and logging, bypassing configured trusted-proxies.	High	High	Resolved
H2: API-key replay protection disabled	<code>apikey_nonce_lifetime</code> $\approx$ 317,000 years	High	Medium	Resolved
H4: Management-API multisig threshold is deploy-config-dependent	If the deployed config is the OpenDAX-default single <code>appplogic</code> signer (1-of-1), one management private-key compromise grants full fund control over the engine (incl. the H7 balance mutators)	High	Medium	Resolved
H5: Missing per-action admin authorization	Comments and <code>phone</code> omit <code>admin_authorize!</code>	High	Medium	Resolved
H6: Deposit credit not idempotent (double-credit)	AASM <code>accept</code> guard evaluated before the row lock.	High	Medium	Resolved
H7: Unlocked P2P/escrow balance mutations (+ H9b value creation)	<code>amount_plus/amount_subtract/swap_amount/bid_operations</code> (<code>account.rb:40-53,224-281</code>) are unlocked read-modify-write, non-atomic across columns	High	Medium	Resolved
H8: Staking admin authorization is a no-op $\rightarrow$ reward-inflation fund minting	<code>Member#authorize!</code> is entirely commented out (staking <code>member.rb:124</code>), so every <code>admin_authorize "Plans",...</code> call is a no-op and <code>/admin/*</code> needs only a logged-in member.	High	Low	Resolved
H9: Staking management signing key committed	<code>staking/config/staking.yml:4</code> commits the RSA key that signs <code>interchange_funds</code> management calls to Peatio.	High	Low	Active
H10: Non-atomic internal (user-to-user) transfer	Receiver credit and sender debit run in separate transactions across an <code>after_commit</code> boundary (<code>withdraw.rb:108-144,570-592</code>).	High	Low	Resolved

Risk Issues

☐ Active
 ☐ New
 ☐ Acknowledged
 ☐ Remediated
 ☐ Resolved

Vulnerability	Description	Risk	Probability	Status
H11: Authenticated SQL injection via <code>Order.market_condition</code>	<code>Order.market_condition</code> (<code>app/models/order.rb:821-832</code>) builds a SQL <code>WHERE</code> array by raw-interpolating request params	High	High	Resolved
H12: Committed OKX exchange API credentials (key + secret + passphrase)	A full OKX API credential set is committed as hardcoded fallback defaults: <code>lib/peatio/okx/api/configuration.rb:17-27</code> and <code>lib/peatio/upstream/okx/client.rb:35-37</code>	High	Medium	Remediated
H13: User-settable <code>is_fake</code> order flag → account/ledger divergence (fee minted)	The market-making <code>is_fake</code> (and <code>confirmed</code>) order flag is exposed as a user-settable parameter on the public order-create path.	High	High	Resolved
H14: Unauthenticated Vault channel! (<code>ssl_verify=false</code> / plaintext transit) → MITM 2FA bypass + API-key-secret disclosure	Barong keeps its secrets in Vault: API-key signing secrets (<code>vault_attribute :secret</code> , <code>app/models/api_key.rb:16</code> , transit engine) and TOTP 2FA seeds/validation (Vault TOTP engine, <code>app/services/totp_service.rb</code>).	High	Medium	Remediated
H15: Unconditional trust of client-supplied <code>CF-Connecting-IP</code> header → source-IP spoof bypasses the per-API-key IP allowlist (remediation regression)	A holder of a stolen but IP-restricted API key defeats the allowlist by sending a spoofed header.	High	Medium	Resolved
M1: Bearer-token signature not verified	Verifier built with nil public key.	Medium	Low	Resolved
M2: Sessions effectively never expire	~1,141-year expiry, refreshed each request; bound only to User-Agent (IP-binding is dead code).	Medium	Medium	Resolved
M3: Non-constant-time HMAC comparison	API-key signature compared with <code>==</code> (timing side-channel).	Medium	Low	Resolved
M4: Full request env logged via debug <code>puts</code>	Debug line dumps every request's headers/cookies/tokens to logs.	Medium	High	Resolved
M5: Default admin seed identity	<code>config/seeds.yml</code> defaults to <code>admin@barong.io</code> ; password from env (nil if unset).	Medium	Medium	Resolved
M6: Weak OTP / verification-code randomness (entropy review)	Every security-sensitive short code uses <code>rand(10_000..99_999)</code> .	Medium	Medium	Resolved
M7: <code>update_columns</code> validation bypass	<code>/resource/users/preference</code> writes via <code>update_columns</code> , skipping validations/callbacks.	Medium	Low	Resolved
M8: <code>generate_code</code> acts on arbitrary email	<code>/resource/phones/generate_code</code> keys off caller-supplied email.	Medium	Medium	Resolved
M9: Broken authz (nil subject)	<code>delete_user</code> calls <code>admin_authorize! :destroy, user</code> before <code>user</code> is assigned (nil).	Medium	Low	Resolved

Risk Issues

☐ Active
 ☐ New
 ☐ Acknowledged
 ☐ Remediated
 ☐ Resolved

Vulnerability	Description	Risk	Probability	Status
M10: API-key secret returned by detail endpoint	<code>GET /resource/api_keys/:id</code> returns plaintext <code>secret/kid</code> (owner-scoped, TOTP-gated).	Medium	Low	Active
M11: No DB negative-balance constraint	Balance mutations use <code>update_columns/update_attribute</code> , bypassing the <code>>=</code> validation.	Medium	Medium	Resolved
M12: State-machine / fund-movement coupling caller-dependent	Fund moves live in AASM <i>after</i> callbacks in their own nested transactions; only the withdraw-create path wraps the transition.	Medium	Low	Remediated
M13: Staking double-claim race on unstake	Unstake reads state without a row lock before transition.	Medium	Low	Resolved
M14: Admin dashboard authorization is client-side only	<code>exchange-castle</code> gates the console on <code>state==='active'</code> (no role check) and a decorative/inverted <code>disableCheck</code> ; every privileged action relies on server <code>admin_authorize!</code> .	Medium	Medium	Resolved
M15: Admin CSRF token in JS-readable cookie	<code>exchange-castle</code> stores the CSRF token in a non-HttpOnly cookie read by JS (<code>getCsrftoken.tsx</code>).	Medium	Low	Active
M16: Frontends missing security headers (admin + user)	Neither <code>exchange-castle/nginx.conf</code> (admin) nor <code>frontend-ui/nginx.conf</code> (user) sets CSP / X-Frame-Options / HSTS / X-Content-Type-Options / Referrer-Policy.	Medium	Medium	Resolved
M17: Swap / bot-transfer double-entry gap	Swaps record Liability + Revenue but no <code>Operations::Asset</code> and bypass <code>validate_accounting_equation</code> (<code>swap.rb:383-413</code>).	Medium	Low	Active
M18: User enumeration on login (timing + response)	Login runs <code>bcrypt</code> only for <i>existing</i> users (non-existent error before <code>authenticate</code>) and returns distinct status/messages for non-existent vs banned vs deleted vs wrong-password.	Medium	High	Resolved
M19: 2FA enable/disable not gated on password	<code>resource/otp.rb:53,89</code> require only a session + a TOTP code, not the account password.	Medium	Medium	Resolved
M20: Realtime (ranger) edge-authorization gaps	Ranger WS does no <code>Origin</code> allow-list (CSWSH risk) and trusts the gateway-injected <code>Authorization</code> header with no secondary binding.	Medium	Low	Active
M21: <code>force_ssl</code> fail-open default (engine + staking)	<code>exchange-base & staking</code> production set <code>config.force_ssl = ENV['FORCE_SECURE_CONNECTION']=='true'</code> .	Medium	Low	Resolved
M22: Caller-supplied Sumsb <code>applicant_id</code> binding without ownership correlation (control shipped-but-unwired)	Authenticated <code>POST /resource/sumsub/applicant_id</code> (<code>documents.rb:124-128</code>) binds a caller-supplied <code>applicant_id</code> to <code>current_user</code> with no <code>externalUserId</code> / user equivalence check.	Medium	Low	Resolved

Risk Issues

☐ Active
 ☐ New
 ☐ Acknowledged
 ☐ Remediated
 ☐ Resolved

Vulnerability	Description	Risk	Probability	Status
M23: Deposit attribution is fallback-free & case-sensitive (no reconciliation)	Inbound deposits credit only via an exact, case-sensitive <code>PaymentAddress.find_by(address: to_address)</code> match (<code>blockchain_service.rb:206,226</code>).	Medium	Medium	Remediated
M24: Admin 2FA-disable lacks step-up re-authentication	An admin can disable any non-superadmin user's 2FA (<code>otp:false</code>) via <code>POST/PUT /admin/users/update</code> with only <code>admin_authorize! :update, User</code> .	Medium	Medium	Resolved
M25: Unsafe reflection: <code>constantize</code> on user input	The CSV-export path builds a class from request params and calls a method on it.	Medium	Low	Resolved
M26: Sensitive parameters logged in cleartext (debug logging + minimal filter)	Production runs at <code>config.log_level = :debug</code> (<code>config/environments/production.rb:95</code>), which logs full request parameters.	Medium	Medium	Resolved
M27: Vulnerable & outdated dependencies (gems + npm)	OWASP A06. Vulnerable dependencies in every repository	Medium	Medium	Active
M28: Password change does not revoke active sessions	The change-password flow (<code>resource/users.rb:285</code> <code>current_user.update(password:)</code>) and the User <code>after_update</code> callbacks do not flush the user's JWT/browser sessions .	Medium	Medium	Resolved
M29: Excessive privilege in operational roles (fund-config not segregated)	The in-handler ability layer grants non-superadmin operational roles control over fund-custody configuration .	Medium	Medium	Remediated
M30: Four inconsistent "exclude this account" predicates	The engine decides "is this account a market-maker/bot/house account to be treated specially" with four independent, non-coinciding predicates .	Medium	Medium	Active
M31: Non-prod Vault secret-at-rest uses a public MD5-derived key (no secret, no IV)	vault-rails encryption is gated on environment.	Medium	Medium	Active
M32: Fireblocks deposit recipient not pinned to the API-verified transaction	The remediated C1 handler re-fetches and overrides only <code>status/amountInfo/feeInfo</code> from the Fireblocks API.	Medium	Low	Active

Risk Issues

☐ Active
 ☐ New
 ☐ Acknowledged
 ☐ Remediated
 ☐ Resolved

Vulnerability	Description	Risk	Probability	Status
M33: Fireblocks webhook signature trust is config-fragile (JWKS URL / V1 legacy key)	<code>lib/fireblocks/webhook_signature_verifier.rb</code> fetches V2 signing keys from <code>FIREBLOCKS_JWKS_URL</code> (env, prod default) and also accepts a V1 legacy path validated by <code>FIREBLOCKS_WEBHOOK_PUBLIC_KEY</code> .	Medium	Low	Resolved
M34: Reward-setting <code>currency_id</code> made mutable post-creation	<code>currency_id</code> is editable on a live reward setting.	Medium	Medium	Remediated
M35: Public JWT signing oracle (<code>/encrypt_token</code>)	Public unauth <code>POST /encrypt_token</code> signs arbitrary caller claims with the prod key and returns a prod-signed RS256 token carrying any <code>address/currency/amount/callback_url</code> + identity claims.	Medium	Low	Resolved
M36: Cross-service fund transfer is non-atomic:	In <code>Account#transfer_asset!</code> the local ledger writes commit first, then an <code>ActiveRecord.after_all_transactions_commit</code> block (<code>app/models/account.rb:226</code>) fires the synchronous peatio transfer.	Medium	Low	Active
M37: Matching executor settles trades against already-cancelled bot orders	The WAIT-state settlement invariant (<code>app/trading/matching/executor.rb:90,94,139-141</code>) was relaxed so the engine executes matches on bot orders in <code>cancel</code> state (<code>fe28d09</code>).	Medium	Low	New
L1: Permissive CORS (*)	<code>api_cors_origins='*'</code>	Low	Low	Active
L2: User Enumeration via Public <code>/status</code>	Public <code>/status</code> uid enumeration	Low	Low	Remediated
L3: Missing-Authz Information Disclosure (<code>/user_report</code> , <code>/abilities</code>)	<code>/user_report</code> & <code>/abilities</code> missing authz	Low	Low	Resolved
L4: Admin Metrics Raw-SQL Interpolation	admin metrics raw-SQL (admin-gated)	Low	Low	Resolved
L5: Hardcoded Activation Code <code>'12345'</code>	hardcoded <code>'12345'</code> activation code (latent)	Low	Low	Resolved
L6: Global Fixed-Key Cache Write	global <code>Rails.cache.write('params',...)</code>	Low	Low	Resolved
L7: Unauthenticated Member Fee-Group Lookup	unauth member fee-group lookup	Low	Low	Resolved
L8: Staking Committed Dev <code>secret_key_base</code> + Hardcoded Worker Secret	staking committed dev <code>secret_key_base</code> + hardcoded worker secret	Low	Low	Resolved
L9: Admin Dashboard Outdated <code>axios</code> + Dead Code	admin dashboard outdated <code>axios ^0.25.0</code> + dead code	Low	Low	Remediated
L10: Password-Reset Missing Rate-Limit / Resend Throttle	password-reset has no rate-limit/resend throttle (CAPTCHA-only)	Low	Low	Resolved
L11: TOTP Reuse Check Commented Out (replay within skew window)	TOTP reuse check commented out → replay within skew window (<code>totp_service.rb:46</code>)	Low	Low	Resolved

Risk Issues

☐ Active
 ☐ New
 ☐ Acknowledged
 ☐ Remediated
 ☐ Resolved

Vulnerability	Description	Risk	Probability	Status
L12: <code>EncryptionService</code> Key Fallback to 'Test' + <code>/refresh</code> Stale-Cache DoS	<code>EncryptionService</code> key falls back to constant 'Test' if <code>SECRET_KEY_BASE</code> unset + <code>/refresh</code> stale-cache <code>NoMethodError</code> DoS	Low	Low	Remediated
L13: Frontend Residual Hardening	outdated <code>axios</code> + <code>npm-audit</code> vulns (both frontends), uncoded-error i18n passthrough, reset-token-in-URL query, tabnabbing on external <code>target="_blank"</code>	Low	Low	Remediated
L14: Ranger Plaintext WS + PII in Debug Logs	ranger plaintext WS (<code>secure:false</code> , TLS at gateway) + private message payloads written to debug logs	Low	Low	Active
L15: Infra Config Hygiene	MaxMind license key in <code>exchange-auth/Dockerfile</code> , <code>VAULT_TOKEN:changeme</code> default, unpinned <code>:latest</code> plugin base images, <code>exchange-base/config/secrets.yml</code> dev key, <code>castle</code> <code>nginx</code> missing <code>server_tokens</code> off, SPA <code>try_files</code> reflecting raw <code>\$request_uri</code> , and a second wildcard CORS layer in Barong (<code>application.rb:25-30</code> , credentials off); plus two secrets confirmed by a <code>gitleaks</code> history scan	Low	Low	Remediated
L16: Unauthenticated Disclosure of Internal Fireblocks Asset Config	<code>GET /api/v2/peatio/public/currencies (+ /networks)</code> returns each network's <code>options.asset_id</code> (Fireblocks asset id) + gas config with no auth	Low	Low	Resolved
L17: Anti-Phishing Code Accepted as Withdrawal Authenticator	the anti-phishing code is accepted as a withdrawal authenticator in place of OTP	Low	Low	Resolved
L18: Referral Share-Reward Credits Unvalidated Recipient	the referral share-reward endpoint credits an unvalidated recipient	Low	Low	Resolved
L19: ReDoS-Vulnerable Signup URL Validation Regex	The signup URL-field check (<code>frontend-ui Schema/signup.tsx:184</code>) uses ... <code>([/\w .-]*)*\/*/?\$</code> , a nested quantifier with catastrophic backtracking; a crafted input hangs the user's own browser (client-side self-DoS)	Low	Low	Resolved
L20: File Upload Validated by Extension Only (no MIME / magic-byte)	File upload validates by extension only	Low	Low	Remediated
L21: Withdrawal Destination Not Re-Validated Between Accept and Signing	<code>workers/amqp/withdraw_coin.rb:125</code> hands <code>withdraw.rid</code> straight to the Fireblocks <code>create_transaction!(toAddress: withdraw.rid)</code> at send time	Low	Low	Remediated
L22: Vault-Rails Fork Logs Decrypted Plaintext (<code>puts request_opts</code>)	<code>transform_decode</code> runs <code>puts request_opts (plugins-vault-rails/lib/vault/rails.rb:167)</code>	Low	Low	Active
L23: Vault-Rails Fork <code>require "pry"</code> in Production Crypto Library	A development/debug console gem shipped into the production secret-handling path	Low	Low	Active

Risk Issues

☐ Active
 ☐ New
 ☐ Acknowledged
 ☐ Remediated
 ☐ Resolved

Vulnerability	Description	Risk	Probability	Status
L24: Committed JWT Key-Pair in Repository (<code>config/rsa-key</code>)	a valid JWT RSA key-pair committed at <code>config/rsa-key</code> .	Low	Low	Active
L25: <code>/verify_access_token</code> Issues Session Cookie on Error Response	A forged-token request returns <code>500</code> yet the response still carries <code>Set-Cookie: _yimmit_session=...</code>	Low	Low	Resolved
L26: Unauthenticated 500 via <code>Integer > nil</code> in <code>store_redis_cache</code>	<code>store_redis_cache (identity/utils.rb)</code> does <code>session_keys.size > options[:allowed_sessions]</code> , and the only caller passing empty options (<code>verify_access_token, data={}</code>) leaves <code>allowed_sessions nil</code>	Low	Low	Resolved
L27: No <code>.dockerignore:</code> <code>.git/secrets</code> baked into images	Rails repos have no <code>.dockerignore</code> and <code>COPY . \$APP_HOME</code> , baking <code>.git</code> (incl. keys removed from the tree) and <code>secrets.yml</code> into image layers.	Low	Medium	Resolved
L28: Remediation Rack::Attack throttles reference a non-existent path & never fire	Two throttles (<code>config/initializers/rack_attack.rb:46,56</code>) key on <code>identity/sessions/otp/ip / identity/sessions/resend_otp/ip</code> — paths that don't exist, so they never match; <code>resend_otp</code> is left with no IP rate limit (<code>319d8fe</code>).	Low	Low	New
L29: Removing RequestDecryptor also dropped non-GET query-string stripping	<code>70a2b20</code> deleted the <code>RequestDecryptor</code> middleware (<code>config/initializers/grape.rb</code>), which also ran <code>remove_query_params!</code> on every non-GET request; query params can now merge into request params on POST/PUT/DELETE.	Low	Low	New
L30: Captcha fail-open default on the operator-console login	<code>captchaType</code> defaults to <code>"none"</code> (<code>src/api/config.tsx:32</code>), so a missing/misconfigured <code>REACT_APP_CAPTCHA_TYPE</code> build var silently drops the Turnstile widget + <code>captcha_response</code> (<code>Login.tsx:214,455</code>) — previously validated unconditionally (<code>76254c4</code>).	Low	Low	Resolved
L31: New <code>/threshold-fee</code> admin route omits the client-side permissions gate	The new <code>ThresholdFee</code> admin route (<code>src/App.tsx:837-841</code>) is registered without the <code>permissions={userRole}</code> prop its sibling sweep routes carry, so it skips even the M14 client-side role redirect (mitigated by force-logout + server-side authz).	Low	Low	Resolved
I1: Path-Prefix Permission Matching	<code>validate_permissions!</code> matches with <code>@path.starts_with?(a.path)</code> (<code>authorize.rb:162</code>).	Info	Info	Remediated
I2: CSRF Enforcement Gated on <code>platform == 'web'</code>	<code>validate_csrf!</code> (<code>authorize.rb:142-156</code>) enforces CSRF only when <code>session[:platform] == 'web'</code> , and the empty-token check (<code>:143</code>) is commented out.	Info	Info	Resolved
I3: Configuration Smells	<code>kyc_provider</code> set twice and ends as <code>''</code> (<code>001_barong.rb:49/54</code>); defaults include <code>event_api_rabbitmq_password='guest'</code> and <code>domain='openware.com'</code> .	Info	Info	Remediated
I4: JWT Codec — Misspelled Algorithm Option & No Algorithm Allowlist	The signing codec (<code>lib/barong/jwt.rb</code>) keys its algorithm option on the misspelled <code>:algorithm</code> and applies no algorithm allowlist on <code>encode</code>	Info	Info	Resolved
I5: Engine Auth Middleware — Fail-Closed Improvement, Reintroduced Prefix Pattern	The deployed <code>auth/middleware.rb</code> is now fail-closed (<code>401 jwt.missing</code> for non-public v2 paths) but reuses the I1 <code>start_with?</code> prefix allowlist and adds <code>/api/v2/management/</code> to a path-based bypass.	Info	Info	Resolved
I6: Incomplete / Dead Payment-Gateway Feature (delete it)	A <code>zfl</code> deposit-link endpoint minting URLs to a non-existent page + public <code>encrypt_token/decrypt_token</code> with no consumer.	Info	Info	Resolved
I7: Full security-audit findings/remediation map committed to the repo (<code>audit-test.js</code>)	A console script <code>frontend-ui/audit-test.js:1-125</code> (<code>c7f8074</code>) committed at HEAD enumerates the audit findings and their remediation status — an internal roadmap leak that should be deleted before the repo is exposed.	Info	Info	New

Remediation Roadmap

Priority	Finding	Action	Effort
Immediate	C2: <code>/verify_access_token</code> takeover	Remove the endpoint or verify the JWT signature (never <code>decode_token</code> for trust).	Small
Immediate	C1: Unauthenticated deposit webhook (theft of funds)	Verify the Fireblocks webhook signature + source IP; re-fetch the transaction from Fireblocks and credit only server-verified amounts after confirmations; idempotency on a trusted key.	Small-Medium
Short-term	M35: <code>/encrypt_token</code> signing oracle	Remove the endpoint; mint confirmation tokens server-side only; do not auto-open sessions from confirmation tokens.	Small
Immediate	C6 / C3 / C4: Committed secrets (org-wide)	Rotate every committed secret. Treat all as burned; purge history; add <code>.dockerignore</code> + pre-commit secret scanning; move to the secrets manager.	Medium
Immediate	C5: Full user-table disclosure	Gate <code>GET resource/users/records/:platform</code> with <code>admin_authorize!</code> (or remove it); never raw <code>present paginate(data)</code> .	Small
Short-term	H5: Authorization gaps	Add <code>admin_authorize!</code> to the four unguarded admin endpoints (<code>comments</code> , <code>admin/phone</code> POST); adopt default-deny on the admin mount.	Small
Short-term	H4: Mgmt-API multisig · L24: dev key	Verify Peatio requires >1 mandatory signer for fund-moving scopes (H4, deploy config). Remove <code>config/rsa-key*</code> from the repo and purge history.	Small
Immediate	M4: Debug <code>puts</code> logging	Remove the <code>puts "...#{@request.env}"</code> line in <code>Barong::Authorize#remote_ip</code> .	Trivial
Short-term	H1: XFF trust	Derive client IP from <code>CF-Connecting-IP</code> (or rightmost-untrusted XFF), not leftmost raw XFF.	Small
Short-term	H2: Nonce window	Set <code>apikey_nonce_lifetime</code> to seconds/minutes; track used nonces to prevent replay.	Small
Medium-term	M1: Bearer verification	Pass the public key to the authenticator; verify the RS256 signature before trusting claims.	Small
Medium-term	M2 / M3 / M5 / L1	Real absolute+idle session lifetimes; constant-time signature compare; enforce a strong non-default admin; tighten CORS origins.	Small

Vulnerabilities **Critical**

C1: Unauthenticated Fireblocks Deposit Webhook → Balance Minting (Theft of Funds)

Vulnerability severity:	Critical	Vulnerability probability:	High	Status:	Resolved
Code-fixed <code>6252ad5</code> by verifying Fireblocks RS512 signature over the raw body, refetching, and enforcing idempotency					

Mid-engagement disclosure. Because this is an actively-exploitable, unauthenticated fund-theft path, C1 was disclosed to Yimmit ahead of this report in the Urgent Notice of **2026-06-05** flagged for immediate mitigation.

The Peatio engine (`exchange-base`) exposes `POST /api/v2/public/fireblock/webhook` (`app/api/v2/public/fireblock.rb:23`) with **no authentication**. The auth middleware only runs when an `Authorization` header is present (`app/api/v2/auth/middleware.rb:8,17-22`): a header-less request skips authentication and is not rejected. There is no Fireblocks signature check, no HMAC, and no source-IP allowlist.

Impact. An attacker mints arbitrary balances with no real on-chain deposit, then trades or withdraws genuine assets — **direct theft from the exchange**. No idempotency prevents repetition: the unique index is defeated by a randomized `txout` (`blockchain_service.rb:232`), and the replay guards key off attacker-controlled `fireblock_id`/addresses (`:202,204`), so unlimited credited rows can be created.

Recommendations:

- **Authenticate the webhook:** verify Fireblocks' request signature (Fireblocks signs webhooks with its key) and restrict to Fireblocks source IPs; reject unsigned requests.
- **Do not trust body amounts.** Re-fetch the transaction from the Fireblocks API (by id) and credit only the server-verified amount/asset/destination after the required confirmations.
- **Idempotency** on a server-trusted immutable key (e.g. confirmed (`blockchain_key`, `txid`, `vout`)), not a random `txout`; reject duplicates.
- Fix the auth middleware so unauthenticated requests to non-public routes are rejected, and treat `/public/*` webhooks as untrusted input by default.

Vulnerabilities **Critical**

C2: Unauthenticated Account Takeover via `/verify_access_token`

Vulnerability severity:	Critical	Vulnerability probability:	Low	Status:	Resolved
Code-fixed <code>319d8fe</code> by <code>decode_token</code> now calls <code>::JWT.decode(token, public_key, true, algorithms:['RS256'])</code> (verification ON, alg pinned).					

The public endpoint `GET .../identity/sessions/verify_access_token` (`sessions.rb:782-788`) decodes a caller-supplied JWT and opens a session for its `email` claim. The codec `decode_token` (`jwt.rb:39-40`) calls `::JWT.decode(token, public_key, false)`: the third argument disables verification, so the public key is ignored and the signature is never checked. The sibling `decode_and_verify` (`jwt.rb:34`) passes `true` with `iss/aud/exp/sub` checks; this endpoint does not use it.

```
get '/verify_access_token' do
  decoded      = codec.decode_token(params[:access_token], keystore.public_key)
  current_user = get_user(decoded[:email])      # user resolved from the UNVERIFIED email claim
  csrf_token   = open_session(current_user, data={}) # full session opened
end

def decode_token(token, public_key)
  # lib/barong/jwt.rb:39
  payload, _ = ::JWT.decode(token, public_key, false) # 3rd arg false: verification OFF, key ignored
end
```

Impact. A forged token with any `email` claim is issued a full session as that account, including administrators, with no password and no 2FA: unauthenticated account takeover across the entire user base and the admin plane. The same flaw makes the cleartext-logged `access_token` (M26) directly forgeable and, with the weak withdraw second factor (L17), reaches fund withdrawal.

Live status (production, 2026-06-17). The signature bypass is confirmed: a token with a garbage signature was accepted and reached `open_session`, where a verifying endpoint would have returned `401` at decode. The takeover does not complete, but only because of two unrelated bugs in this same handler. It calls `open_session` with empty options, so `store_redis_cache` compares the session count against a nil `allowed_sessions` and raises (L26), returning `500` before the session commits; and the cookie set on that `500` carries `platform=nil`, which fails the CSRF gate (`authorize.rb:144`). It is blocked by accident, not by any control: this is the only one of six `open_session` call-sites that passes empty options, and fixing either bug turns the confirmed bypass into full takeover.

Recommendations:

- Remove the endpoint. If the flow is genuinely needed, verify the JWT signature (RS256 against the production public key) plus `iss/aud/exp` before resolving a user or opening a session.
- Never use `decode_token` (verification off) for any trust decision; reserve it for non-security parsing.
- Never derive session identity from an unverified `email` claim; bind sessions to a server-issued token.

Vulnerabilities

Critical

C3: Hardcoded Jenkins Admin Credentials

Vulnerability severity:	Critical	Vulnerability probability:	Medium	Status:	Remediated
Remediated at 6180878 by Hardcoded Jenkins URL/username/API token/build AUTH token externalized to ENV.					
! Outstanding to resolve: Purge the committed Jenkins tokens from git history — the prod re-test (2026-07-17) confirmed the deployed JENKINS_TOKEN is already rotated.					

exchange-base/lib/restart_service/client.rb:9-12 commits a Jenkins admin API token, a build trigger token, and the internal Jenkins IP:

```
JENKINS_URL = 'http://10.0.138.220:8080/'
USERNAME    = 'admin'
API_TOKEN   = '11a2c5774de77ecb5064f71ee0313b60c9'
AUTH_TOKEN  = '11b03bd16216b5a497c7f0c70d8b29e803'
```

Per docs/Exchange CI_CD Doc.pdf, Jenkins builds and pushes Docker images to AWS ECR, which ArgoCD syncs to the production EKS cluster. The admin API token permits arbitrary job execution, arbitrary image build/push, and production code execution. It compounds the leaked GitHub PAT (C6): both grant write access to the same image-supply pipeline. Internal-IP reachability bounds external exploitability; the exposed credential (insider, repo/archive copies, or any network foothold) is the issue. The newer jenkins_job_trigger.rb correctly uses ENV['JENKINS_TOKEN'].

Recommendations:

- Rotate the Jenkins admin token and build tokens; remove the legacy client; move all CI creds to the secrets manager; restrict Jenkins to least-privilege per-job tokens and network-segment it.

C4: Committed Event-API RSA Private Key

Severity:	Critical	Probability:	Low	Status:	Remediated
Remediated at 6180878 by Committed RSA private-key default replaced with nil at both occurrences.					
! Outstanding to resolve: Purge the committed 2048-bit key from git history — the prod re-test confirmed the deployed Event-API key is rotated (sha256 e9ef2683... ≠ committed).					

exchange-base/config/application.yml:133 commits a full 2048-bit -----BEGIN RSA PRIVATE KEY----- as the default for EVENT_API_JWT_PRIVATE_KEY, the key that signs inter-service Event-API JWTs (consumed at config/initializers/event_api.rb:140). If production does not override this default, an attacker can **forge Event-API messages between services**; regardless, it is a committed signing key and must be treated as compromised.

Systemic note. C3/C4 join an unusually large set of committed secrets across the repos (rsa-key, GitHub PAT, staking management key, Jenkins token, Event-API key, MaxMind key). Recommend a one-time org-wide secret rotation + history purge + pre-commit secret scanning, treating every committed key as burned.

Recommendations:

- Rotate the key
- Purge git history
- Implement secrets scanning as a CI gate

Vulnerabilities

Critical

C5: Full User-Table Disclosure to Any Authenticated User

Severity:	Critical	Probability:	High	Status:	Resolved
mass-dump endpoint removed, dd7e6d1/c81bb3a; password_digest excepted + peppered, d453058; verify records/* 404 + BARONG_PEPPER set, live					

GET /resource/users/records/:platform (app/api/v2/resource/users.rb:163) is a resource-namespace endpoint, reachable by **any authenticated member**, that returns the entire users table with **default (raw) serialization, no admin_authorize! and no current_user scoping**:

```
get '/records/:platform' do
  data = User.where(platform: params[:platform])
  error!({ errors: ['resource.user.no_records'] }, 422) unless data.present?
  present paginate(data)
end
```

Live-confirmed (2026-06-17). A plain member session (self-owned test account; role: member verified via /resource/users/me) called records/exchange (exchange is the schema default for users.platform, db/schema.rb:338) and received HTTP 200 with Total: 182, the **entire user base**, enumerable page-by-page by any one of those members. Returned rows were confirmed to be other users (uid ≠ the tester's).

- Leaked field set (raw users columns, confirmed present).** Default serialization returns the raw row, including:
- password_digest: the bcrypt password hash for every user (offline-crackable).
 - code, code_expire_at, verification_code, reactive_code_expire: the **live email/SMS second-factor one-time codes**. This is the exact column the login OTP is checked against (sessions.rb:244: params[:code] == user.code).
 - Full PII / KYC linkage:** email, first_name, middle_name, last_name, phone_number, username, user_time_zone, sumsub_applicant_id (KYC-provider linkage), referral_id, level, role, state, metadata, data.

Cracking analysis. bcrypt is salted, adaptive and GPU-hostile, but it makes offline guessing expensive, not impossible, and C5 makes it the **last and only** barrier to every account. The salt and cost factor are stored inside each digest (`$2a$<cost>$<salt><hash>`), so the leak hands an attacker everything needed to verify guesses **offline on their own GPUs**, with none of the rate-limit, lockout, 2FA prompt or logging the server imposes. Per-user salts stop precomputation / rainbow tables and batch-cracking all 182 at once, but do **not** stop guessing each hash: weak or reused passwords fall in seconds-to-minutes to standard breach wordlists (hashcat mode `3200`). The co-leaked `email`/name/phone also let an attacker skip cracking via **credential-stuffing** (reused passwords from other breaches) or **targeted phishing**.

Recommendations:

Add `admin_authorize!` (or remove the endpoint outright); if a member-facing variant is needed, scope to `current_user` and present an **explicit minimal entity**, never raw `present paginate(data)`, excluding `password_digest`, `code/verification_code`, `otp` and all KYC fields.

- Treat the exposure as a potential breach: given the live confirmation, assess a **forced password reset + rotation/invalidation of in-flight login codes**, and review access logs for prior unauthorized reads of this route.
- **Keep bcrypt, don't expose it:** ensure `password_digest` never enters any serializer/entity; as defense-in-depth raise the bcrypt cost factor and add a server-side pepper / HMAC key stored *outside* the database, so a serialization or DB leak alone is useless without it.
- **Scope (namespace audited 2026-06-17):** `records/:platform` is the **only resource:** endpoint returning a raw, unscoped `User`; all other User-returning endpoints use the curated `UserWithFullInfo` entity or are ownership-gated (see P30), so this fix is a single route. The same raw/opt-out-serialization root cause recurs as **M8** (`/generate_code` on arbitrary `user_email`) and **M10** (api-key detail returns `secret`); fix those alongside.

Vulnerabilities **Critical**

C6: Live Committed Admin GitHub PAT

Severity:	Critical	Probability:	High	Status:	Active
Remediation was attempted at 319d8fe by removing the inline PAT from the Gemfile / Gemfile.lock git URLs (no github_pat in the HEAD tree).					
! Outstanding to resolve: the committed PAT was CONFIRMED STILL LIVE (GET api.github.com/user → 200, 2026-07-17); the 2026-08-04 re-test confirms the working tree stays clean but the token remains in git history and its revocation is unverified. Revoke on GitHub, forensic-audit the four private repos, then purge history.					

exchange-auth/Gemfile (lines 66 & 94) and Gemfile.lock embed a fine-grained GitHub PAT (github_pat_11B7S3...) in the git: URLs used to fetch private yimmitOrg plugins at bundle time. It has been present **since the initial commit** (in git history), so it is recoverable by anyone with repo, clone, or archive access.

Live-confirmed (2026-06-17). GET https://api.github.com/user with the committed token returned HTTP 200; the token is **live**, owned by the service account securityhub-lang (id 266711465; an empty, purpose-built CI bot created 2026-03-09). The per-repo permissions object reports admin: true, push: true, and GET /user/repos enumerates **four** private repos the token can reach: plugins-peatio, plugins-vault-rails, plugins-sumsub-ruby-sdk, plugins-recaptcha.

All four repos are **compiled into the production apps** at build time: plugins-vault-rails is the **secret-encryption layer**, plugins-peatio the engine, plugins-sumsub KYC. push/admin to them injects code into the next production build, giving **arbitrary code execution in production + secret exfiltration**.

Recommendations:

- **Revoke the PAT on securityhub-lang immediately** (one token reaches all four repos); do not merely delete the Gemfile lines (it is in history and already cloneable).
- **Forensic-audit the four repos** (commits / webhooks / deploy keys / collaborators / branch protection / Actions) for tampering before trusting the next build; rebuild from a known-good state if anything is off.
- Re-provision CI via a **scoped deploy key or GitHub App** with read-only Contents on only the needed repos, never an admin PAT inline in a Gemfile; add secret scanning + a history purge.

Vulnerabilities High

H1: Client-IP Spoofing via X-Forwarded-For at the Gateway

Vulnerability severity:	High	Vulnerability probability:	High	Status:	Resolved
Remediated at 77a2c5b / 9fc9911 by resolving the client IP via the trusted-proxy-aware <code>remote_ip</code> (<code>authorize.rb</code> & <code>api/v2/utils.rb</code>), closing the leftmost-XFF spoof.					
The leftmost-XFF fix holds, and the regression it introduced (H15 , unconditional CF-Connecting-IP trust) is now also fixed.					

Both `Barong::Authorize#remote_ip` (`lib/barong/authorize.rb:210`) and `AuthorizeController#remote_ip` (`app/controllers/authorize_controller.rb:111`) take the **leftmost** value of the raw `HTTP_X_FORWARDED_FOR` header:

```
ip = @request.env['HTTP_X_FORWARDED_FOR'].split(',').first
```

This bypasses the configured Rails `action_dispatch.trusted_proxies` hardening (`config/initializers/trusted_proxies.rb`). The gateway is `cloudflare` (`001_barong.rb:56`), which **appends** the real client IP to XFF rather than replacing it, so an attacker-supplied leftmost value wins. The resulting `remote_ip` feeds:

Impact. The OTP lockout (`FailedLoginAttempt`) is keyed on this same spoofable `remote_ip` + `user_id`, so rotating `X-Forwarded-For` resets the per-IP attempt counter, defeating the login-OTP lockout and making the 5-digit second factor brute-forceable. XFF trust here directly breaks a 2FA control. Concrete exploit in M6.

Recommendations:

- Derive the client IP from `CF-Connecting-IP` (Cloudflare) or the rightmost-untrusted XFF entry, consistent with `trusted_proxies`.
- Reject or normalize requests where XFF disagrees with the connecting peer at the edge.

H2: API-Key Replay Protection Effectively Disabled

Vulnerability severity:	High	Vulnerability probability:	Medium	Status:	Resolved
Code-fixed 319d8fe by Nonce lifetime cut 1e16ms->10000ms AND per-(kid,nonce) Redis SETNX dedup (<code>Rails.cache.write unless_exist</code>) rejects any replay.					

`apikey_nonce_lifetime` is set to `10_000_000_000_000_000` ms ($\approx$ **317,000 years**) in `001_barong.rb:55`. The nonce/timestamp-window check (`authorize.rb:109-113`) accepts any nonce, so a captured signed API request is replayable indefinitely.

Impact. Combined with H1, a captured trading or withdrawal API call can be replayed from any IP.

Recommendations:

- Set the nonce lifetime to a small window (seconds/minutes) and enforce it.
- Track consumed nonces per key to reject replays inside the window.

Vulnerabilities High

H4: Management-API Multisig Threshold Is Caller-Enforced

Vulnerability severity:	High	Vulnerability probability:	Medium	Status:	Resolved
Code-fixed 6180878 / 8220c45 by management_api_v1.yml ships a populated keychain and a scopes block where write_withdraws/write_transfers/write_deposits/write_operations(_staking) each set mandatory_signers to two independent keys.					

The `rubykube/jwt-multisig` scheme does **not** enforce an N-of-M threshold; the *consumer* declares required signers in `config/management_api*.yaml`. In this archive, all of `management_api{,_v1,_v3,_v4,_v5}.yaml` ship with empty `keychain: {}` / `scopes: {}`, populated at deploy. The number of signatures Peatio requires to honor a fund-moving command is therefore not verifiable from source. [Fixed since audit — Resolved: `config/management_api_v1.yaml` now ships a **populated** `keychain (:22)` and a `scopes` block with two `mandatory_signers` per fund-moving scope — `['barong', 'applogic']` on `write_withdraws/transfers/deposits (:85-94)` and `['staking', 'staking_cosign']` on `interchange_funds (:111-112)`; the threshold is now **verifiable in-repo** ([6180878/8220c45](#)).]

The consumer side is confirmed in the Peatio engine (`exchange-base`): the management JWT middleware verifies via `JWT::Multisig.verify_jwt` against per-scope `permitted_signers` (`api/v2/management/jwt_authentication_middleware.rb`), and the fund-moving `write_withdraws` scope drives `withdraw create/action` (`api/v2/management/withdraws.rb`). Its `config/management_api_v1.yaml` also ships with empty `keychain: {}` / `scopes: {}`. The entire `signer_policy` for fund-moving management actions is operator-supplied and absent from every repo. [Fixed since audit — Resolved: the deployed-repo `config/management_api_v1.yaml:85-115` now defines two `mandatory_signers` per fund scope, and `jwt_authentication_middleware.rb` computes `missing_signers` and fails **closed** (401 "Not enough signatures") — the `signer_policy` is present in the repo, not absent.]

Recommendations:

- Obtain the *deployed* `management_api*.yaml` from Peatio/applogic and confirm `mandatory_signers` requires more than one independent key for fund-moving scopes. [Fixed since audit — Resolved in-repo: `config/management_api_v1.yaml` already sets two `mandatory_signers` per fund-moving scope, so this is now verifiable from source. Narrow residual: the `jwt-multisig` gem's internal `verify_jwt` is not vendored, and custodial independence of the two keys (`barong` vs `applogic`) is operational, not code-visible.]
- Verify `verify_jti`, `verify_aud`, and `exp_leeway` are set appropriately.

H5: Missing Per-Action Authorization on Admin Endpoints

Severity:	High	Probability:	Medium	Status:	Resolved
Code-fixed 319d8fe by All cited endpoints gained the missing per-action CanCan check via <code>admin_authorize!</code> , which raises 401 on <code>CanCan::AccessDenied</code> .					

Two `app/api/v2/admin/*` endpoints omit the `admin_authorize!` (CanCan `AdminAbility`) call that every sibling endpoint uses. The gateway path-rules (`config/seeds.yaml`) grant the `admin` role `admin`.

- `POST /admin/phone` (`admin/phone.rb:32`): wipes a user's phones and creates a **pre-verified** phone (`validated_at: Time.now`) for any uid — a recovery / SMS-2FA vector.
- `PUT & DELETE /admin/users/comments` (`admin/users.rb:993, 1012`): `Comment.find(params[:id])` global lookup by sequential id edits/destroys any user's audit comment (IDOR).

Impact. Without per-action authorization, an admin-tier principal can set a **pre-verified phone** on an arbitrary uid (a recovery / SMS-2FA vector) and edit or destroy any user's audit comments.

Recommendations:

- Add `admin_authorize!` with the correct ability/subject to each endpoint; adopt a default-deny posture on the admin mount so a missing check fails closed.

Vulnerabilities High

H14: Unauthenticated Vault Channel: `ssl_verify = false` / Plaintext Transit

Severity:	High	Probability:	Medium	Status:	Remediated
Remediated at <code>319d8fe</code> / <code>6180878</code> / <code>47505c0</code> by removing the hardcoded <code>ssl_verify = false</code> (now <code>true</code> / ENV-default <code>true</code>).					
! Outstanding to resolve: the deployed Vault address is still <code>http://</code> on both channels — engine <code>VAULT_URL</code> and Barong <code>BARONG_VAULT_ADDRESS</code> (the 2FA-critical one) — so <code>ssl_verify=true</code> is moot and the plaintext-transit MITM path is confirmed live in prod . Move Vault to <code>https://</code> and network-isolate the channel.					

Barong stores sensitive secrets in HashiCorp Vault and reaches Vault over a channel that performs **no certificate verification** (and, by default, **no transport encryption**):

- **API-key signing secrets:** `app/models/api_key.rb:16` `vault_attribute :secret` encrypts/decrypts via Vault's *transit* engine on every read/write.
- **TOTP 2FA seeds & validation:** `app/services/totp_service.rb` generates seeds in Vault's *TOTP* engine and validates codes by asking Vault.

The client config disables verification and defaults to cleartext HTTP:

```
# config/initializers/vault.rb
Vault::Rails.configure do |config|
  config.enabled      = Rails.env.production?
  config.address      = Barong::App.config.vault_address
  config.token        = Barong::App.config.vault_token
  config.ssl_verify = false
end
```

2FA bypass. Validation trusts whatever Vault returns, `totp_service.rb:44`:

```
def validate?(uid, code)
  # ...
  write_data(totp_code_key(uid), code: code_str).data[:valid]
end
```

An attacker on the Barong / Vault path (e.g. a foothold in the namespace/cluster where Barong reaches the Vault service) can forge a `{ valid: true }` response to **satisfy 2FA for any account**, and can observe submitted TOTP codes in flight. The same uncrypted channel carries the API-key secret transit round-trips, so the attacker also **harvests plaintext API-key signing secrets**.

Recommendations:

- Set `config.ssl_verify = true` and use `https://` for the Vault address with a pinned CA bundle.
- Mutually authenticate the Barong/Vault connection (TLS client certificate or, preferably, Vault **AppRole** with a short-lived token instead of a static `VAULT_TOKEN`).
- Treat all existing API-key signing secrets and TOTP seeds as exposed: rotate API keys and force TOTP re-enrollment after the channel is hardened.
- Verify the production `VAULT_ADDR` scheme (`http` vs `https`).

Vulnerabilities High

H15: Unconditional Trust of Client-Supplied CF-Connecting-IP Header

Severity:	High	Probability:	Medium	Status:	Resolved
The CF-Connecting-IP short-circuit was deleted; client IP now resolves only via the trusted-proxy-aware remote_ip. Residual: HTTP_TRUE_CLIENT_IP honored only when gateway=='akamai' (config-gated).					

The security-hardening remediation (319d8fe) rewrote Barong’s client-IP resolver to trust the CF-Connecting-IP request header **unconditionally**. It is now returned before any trusted-proxy validation and regardless of the configured gateway, so any caller can set it to an arbitrary value:

```
# lib/barong/authorize.rb:221
def remote_ip
  cf_ip = @request.env['HTTP_CF_CONNECTING_IP']
  return cf_ip if cf_ip.present?

  if Barong::App.config.gateway == 'akamai'
    true_client_ip = @request.env['HTTP_TRUE_CLIENT_IP']
    return true_client_ip if true_client_ip.present?
  end

  ip = ActionDispatch::Request.new(@request.env).remote_ip
  ip = '127.0.0.1' if ip.blank?
  ip
end
```

This spoofable value is the identity used by the controls meant to contain stolen credentials:

- **Per-API-key IP allowlist:** lib/barong/authorize.rb:137 calls current_api_key.ip_allowed?(remote_ip), an exact-match check (app/models/api_key.rb:74-78, ips.include?(ip)). A holder of a stolen but IP-restricted API key defeats the allowlist simply by sending CF-Connecting-IP: <whitelisted-ip>.
- **IP-based device trust & audit trail:** the same value feeds user.trust(ip:) (app/api/v2/identity/sessions.rb:70,214), failed-login attribution, and activity logging. Device-trust decisions can be manipulated and audit-log source IPs forged with no API key at all.

Confirmed live in production (2026-07-17). The origin is reachable without traversing Cloudflare. The internet-facing ALB (k8s-ingressn-ingress-f96cfa5341, hosts app.yimmit.com) has security group sg-00fe6761b92c20d45 with inbound 0.0.0.0/0 on ports 80/443 and **no WAF associated**. An attacker connects straight to the origin and supplies any CF-Connecting-IP; nothing enforces that the header came from a Cloudflare edge.

Recommendations:

- Do not trust CF-Connecting-IP unconditionally. Honour it **only** when the immediate peer (request.remote_addr) is a verified Cloudflare edge IP; otherwise fall back to the trusted-proxy-aware ActionDispatch remote_ip.
- Restrict the origin ALB security group to Cloudflare IP ranges (and/or enforce Cloudflare Authenticated Origin Pull or a shared-secret header) so the origin cannot be reached off-Cloudflare.
- Use one consistent, trusted-proxy-aware IP resolver across the app — align Barong::Authorize#remote_ip with the already-hardened Rack::Attack resolver.
- Treat API-key IP-allowlist enforcement and device-trust records created while this was live as unreliable; review activity logs for forged source IPs.

Vulnerabilities

Medium

M1: Bearer-Token Signature Not Verified

Severity:	Medium	Probability:	Low	Status:	Resolved
authorize.rb now calls authenticate! -> decode_and_verify_token (JWT.decode(..., true, algorithms:['RS256'])); the non-verifying .decode is no longer invoked and a rescue fails closed to 401.					

In `bearer_token_owner` (`lib/barong/authorize.rb:60-77`) the verifier is `Peatio::Auth::JWTAuthenticator.new(@keypub)`, but `@keypub` is never assigned (`nil`, `authorize.rb:267`), and the token is processed via the custom `.decode` (`authorize.rb:63`).

The `peatio-core` fork (`urkishan/peatio-core`, branch `feature/added-decode-method`) adds a `decode` method that decodes without verifying the signature, unlike the gem's secure path:

```
# lib/peatio/auth/jwt_authenticator.rb
def decode(token)
  payload, _ = JWT.decode(token, @public_key, false, @verify_options)
end
def decode_and_verify_token(token)
  payload, _ = JWT.decode(token, @public_key, true, @verify_options)
end
```

The signature check is absent on Barong's bearer path. The only remaining barrier is the server-side Redis gate `Rails.cache.exist?("#{ruuid}_#{uid}")` (`authorize.rb:67`), both values taken from the unauthenticated token, which an attacker cannot satisfy for another user without a valid or predictable `ruuid/uid` pair.

Impact. The cryptographic control on the bearer path is missing; the cache gate alone protects accounts.

Recommendations:

Call `decode_and_verify / verify=true`

M2: Sessions Effectively Never Expire

Severity:	Medium	Probability:	Medium	Status:	Resolved
Code-fixed 319d8fe by 30-min idle + 8-hour absolute session lifetime now enforced.					

`session_expire_time = 36_000_000_000 s` ($\approx$ **1,141 years**, `001_barong.rb:48`), and `validate_session!` (`authorize.rb:79-91`) refreshes `expire_time` on every request, so a session never ages out. Session validation checks **only User-Agent equality** (`authorize.rb:80`); the IP-binding helper `find_ip` (`authorize.rb:93-100`) is dead code, never called. A stolen session cookie works from any IP indefinitely once the User-Agent matches.

Recommendations:

- Enforce absolute and idle session lifetimes; treat User-Agent equality as non-security; bind sessions to a stable, server-derived client signal.

Vulnerabilities Medium

M3: Non-Constant-Time HMAC Signature Comparison

Severity:	Medium	Probability:	Low	Status:	Resolved
Code-fixed 319d8fe by ActiveSupport::SecurityUtils.secure_compare (constant-time).					

APIKeysVerifier#verify_hmac_payload? ([app/services/api_keys_verifier.rb](#)) compares `true_signature == @signature` with `==`, whose runtime varies with input, leaking signature bytes via timing.

Impact. An attacker can recover a valid HMAC signature through timing measurement.

Recommendations:

- Use `ActiveSupport::SecurityUtils.secure_compare` for signature comparison.

M4: Full Request Environment Logged via Debug puts

Severity:	Medium	Probability:	High	Status:	Resolved
the <code>puts @request.env</code> debug line was deleted from <code>Barong::Authorize#remote_ip</code> in the IP-trust rewrite (77a2c5b); the method now logs only the resolved IP via <code>Rails.logger.debug</code> (authorize.rb:215) — no Rack env, headers, or tokens reach the logs					

`Barong::Authorize#remote_ip` contains a debug line ([authorize.rb:209](#)): `puts "====...remote_ip #{@request.env}"`. This prints the entire Rack env on every authenticated request (`Authorization` bearer, `Cookie`, `X-CSRF-Token`, `API-key` headers) to stdout/container logs.

Recommendations:

- Remove the `puts`; ensure request env / secrets are never logged (review [filter_parameter_logging](#) coverage, see M26).

Vulnerabilities Medium

M5: Default Admin Seed Identity

Severity:	Medium	Probability:	Medium	Status:	Resolved
Code-fixed 319d8fe by both default admin email/password fallbacks removed; seeding now uses single-arg ENV.fetch and fails closed when unset					

config/seeds.yml defaults the admin email to admin@barong.io with the password from ENV['ADMIN_PASSWORD'] (nil if unset). The publicly-circulated legacy default password for Barong installs is Qwerty123.

Impact. An attacker who knows the default identity can authenticate as admin if the deployment retains the seeded email and a known or empty password.

Recommendations:

- Seed a non-default admin email and a strong ADMIN_PASSWORD with 2FA enforced. Verify admin@barong.io does not authenticate on the live admin login.

M6: Weak OTP / Verification-Code Randomness (entropy review)

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Resolved
6-digit <code>SecureRandom</code> OTP (<code>OtpCode.generate, lib/barong/otp_code.rb:5</code>), <code>5b91fef</code> ; the spoofable-XFF lockout-reset bypass (M6 + H1 chain) closed by trusted-proxy <code>remote_ip</code> in <code>utils.rb, 77a2c5b/9fc9911</code>					

We enumerated **every** token/code generator and graded it on bit-strength and CSPRNG use. The long-lived tokens are sound. The failure is concentrated in the short security codes, which all use the **non-cryptographic** `Kernel#rand` over a 5-digit space.

Generator	Protects	Entropy	CSPRNG?	Verdict
Session / access / refresh JWT	session auth	RS256-signed + <code>jti</code>	n/a	Sound
<code>device_token (hex(32))</code>	device binding	128-bit	yes	Sound
API-key <code>secret/kid (hex(32))</code>	API auth	128-bit	yes	Sound
<code>reset_token, csrf_token, jti (hex(10)); admin whitelink (SHA256*hex(10))</code>	password reset / CSRF / JWT id / allowlist link	80-bit	yes	Sound
service-account <code>secret, doc identifier (hex(16))</code>	service auth / doc id	64-bit	yes	Sound
login OTP (<code>sessions.rb:461</code>), SMS code (<code>phone.rb:67</code>), email activation (<code>users.rb:36</code>), reactivation (<code>:286</code>), verification (<code>resource/users.rb:211</code>)	email/SMS 2nd factor & account verification	~16.6-bit (5-digit, 90k)	no, <code>Kernel#rand</code>	Finding

The bottom row is M6: a 2nd-factor / verification code that is both **brute-forceable** (90k space, worsened by the missing reset rate-limit, L10) and **predictable** (Mersenne-Twister state, not `SecureRandom`). The login second factor is a constant 5-digit `code = rand(10_000..99_999)`, and the OTP lockout that should bound guessing is keyed on (`ip_address, user_id`) (`FailedLoginAttempt.find_or_initialize_by, sessions.rb:222`; UNIQUE index (`user_id, ip_address`)) where `ip_address` is the spoofable leftmost `X-Forwarded-For` (`utils.rb:10, H1`). A wrong guess does *not* regenerate `user.code` (`sessions.rb:244-247`). An attacker who already holds the password (reused, phished, or cracked via the C5 hash leak) rotates `X-Forwarded-For` every `otp_attempts` tries (chain: M6 + H1). The only remaining backstop is an edge rate-limit by true source IP, which is not version-controlled (cf. M16).

Recommendations:

- Generate codes with `SecureRandom`, ≥6 digits, with expiry and attempt limits.

Vulnerabilities Medium

M7: `update_columns` Validation Bypass

Vulnerability severity:	Medium	Vulnerability probability:	Low	Status:	Resolved
Code-fixed <code>319d8fe</code> by POST <code>/resource/users/preference</code> now uses <code>update</code> (validations + callbacks run) instead of <code>update_columns</code> .					

POST `/resource/users/preference` writes via `current_user.update_columns(declared(params,...))` (`resource/users.rb:62`). The param block constrains the fields (no privilege escalation), but `update_columns` skips validations and callbacks. Self-only.

Recommendations:

- Use `update/assign_attributes+save` so validations and callbacks run.

M8: `generate_code` Acts on Arbitrary Email

Vulnerability severity:	Medium	Vulnerability probability:	Medium	Status:	Resolved
Code-fixed <code>319d8fe</code> / <code>f77740f</code> by Arbitrary-account phone-change trigger closed: handler no longer accepts or resolves a caller-supplied email.					

POST `/resource/phones/generate_code` reassigns `current_user = User.find_by_email(params[:user_email])` (`resource/phones.rb:168`), so an authenticated caller triggers a phone-change token for an arbitrary account; differential responses enable email enumeration.

Recommendations:

- Always operate on the JWT-resolved `current_user`; never key off a caller-supplied `user_email`.

M9: Broken Authorization (`nil` subject)

Vulnerability severity:	Medium	Vulnerability probability:	Low	Status:	Resolved
Code-fixed <code>319d8fe</code> by <code>admin_authorize! :destroy, User (class)</code> resolved before loading subject.					

DELETE `/admin/users/delete_user` calls `admin_authorize! :destroy, user` before `user` is assigned on the next line (`admin/users.rb:1078-1079`); the ability check receives `nil`, failing open or dead depending on CanCan's `nil` handling.

Recommendations:

- Resolve the subject first; pass the loaded `user` instance to `admin_authorize! :destroy`.

Vulnerabilities Medium

M10: API-Key Secret Returned by Detail Endpoint

Vulnerability severity:	Medium	Vulnerability probability:	Low	Status:	Active
-------------------------	--------	----------------------------	-----	---------	--------

GET /resource/api_keys/:id (resource/api_keys.rb:144) returns the plaintext `secret/kid` in the body, while the list/create/update paths redact them. Owner-scoped and TOTP-gated (not IDOR), but a secret-bearing response. **Root cause:** `Entities::APIKey` exposes `secret/kid` **by default**, so safety depends on every caller remembering **except: %i[secret kid]**; the detail route forgot it. Flip the default: never expose `secret` from the shared entity; surface it only via an explicit one-time create-response shape (same opt-out-serialization root cause as C5).

Recommendations:

- Never return the `secret` after creation (transport/log hygiene). Flip the entity default: never expose `secret/kid` from the shared `Entities::APIKey`; surface the secret only via an explicit one-time create-response shape.

M22: Caller-Supplied Sumsb `applicant_id` Binding Without Ownership Correlation

Severity:	Medium	Probability:	Low	Status:	Resolved
-----------	--------	--------------	-----	---------	----------

Code-fixed `319d8fe` / `3b8db79` by Applicant binding now correlates ownership (fails closed on nil), and the webhook digest is pinned to HMAC-SHA256 with constant-time compare.

The authenticated POST /api/v2/barong/resource/sumsub/applicant_id (resource/documents.rb:124-128, behind JWTAuthenticator) binds a caller-supplied `params[:applicant_id]` to `current_user` with **no ownership check**: `SumsbKycService` fetches the applicant with `merchant-level` credentials, reads only `response["levelName"]`, and never compares the applicant's real `externalUserId` to `current_user.uid` (`sumsub_kyc_service.rb:40-66`). A genuine identity control (`registered_number_match_with_sumsub?` (:17-27)) exists but is **dead code**, never wired into the bind or the `reviewStatus` promotion path.

Ownership is enforced only *incidentally*: `applicant_id` is the string PRIMARY KEY of `sumsub_applicants`, so submitting a victim's already-recorded id collides on INSERT and raises `RecordNotUnique` which is caught only by `rescue_from :all` as an HTTP 500, so a DB constraint, not an authorization decision, is the barrier. The residual window is a victim applicant present in Sumsb but not yet persisted locally; a successful bind only attaches an `'initiated'` label, and promotion to `verified` stays gated on Sumsb's signed webhook `reviewStatus`. Separately, that webhook checks `X-Payload-Digest` with non-constant-time `==` (`public/general.rb:83`, same as M3) and leaves a debug `puts digest` at :81. The KYC analog of C1.

Recommendations:

- In `save_applicant` / the endpoint, fetch the applicant's real `externalUserId` from Sumsb and require it to equal `current_user.uid + '-' + level` before persisting (reject on mismatch); and/or wire `registered_number_match_with_sumsub?` into the bind/promote path so identity is correlated rather than relying on PK collisions.
- Do not use `find_or_create_by(id: caller_value)` as an authorization boundary; rescue `ActiveRecord::RecordNotUnique` explicitly so a cross-account attempt fails closed (403/409) instead of leaking an HTTP 500.
- Replace the webhook `==` digest check with `ActiveSupport::SecurityUtils.secure_compare` (constant-time) and remove the debug `puts digest` (:81).

Vulnerabilities Medium

M26: Sensitive Parameters Logged in Cleartext (debug logging + minimal filter)

Severity:	Medium	Probability:	Medium	Status:	Resolved
Remediated at 319d8fe by Replayable secrets are now redacted even at debug level via an extended filter list, but the primary recommendation to lower the production log level was not applied.					
Production log_level lowered to warn, so the request-params hash is no longer emitted; filter list expanded. Residual (defense-in-depth): uid/ruuid unfiltered if debug is force-enabled.					

Production runs at `config.log_level = :debug` (`config/environments/production.rb:95`), which logs the full `Parameters: {...}` hash for every request. Parameter filtering covers only two keys: `Rails.application.config.filter_parameters += %i[password upload]` (`config/initializers/filter_parameter_logging.rb:4`; Rails partial-matches, so password variants are caught but nothing else is). The following are therefore written to production logs in cleartext (CWE-532): login OTP / 2FA codes (`code, otp_code: sessions.rb:167,200; users.rb:230,409`), the unverified-decode `access_token` from the C2 endpoint (`sessions.rb:777`), confirmation/reset `tokens` (`users.rb:333,452`), API secrets, and the withdrawal anti-phishing code. This is the systemic version of M4 and amplifies M1: a logged `ruuid+uid` pair defeats the sole Redis cache-existence gate, and logged OTP/reset tokens are replayable within their windows.

Impact. Anyone with log or aggregator access can replay OTPs, reset tokens, and bearer credentials, and read API secrets in cleartext.

Recommendations:

- Set the production log level to `:info` or `:warn` (the file ships the commented alternative `ENV.fetch('LOG_LEVEL','warn')` at :99) so full request parameters are not logged.
- Extend `filter_parameters` to all sensitive keys: `otp, code, otp_code, token, access_token, refresh_token, reset_token, confirmation_token, secret, api_key, secret_key, pin, apc_code, anti_phishing_code`.
- Audit existing log retention/aggregation for already-captured tokens/OTPs and rotate where warranted.

Vulnerabilities Medium

M31: Non-Prod Vault Secret-at-Rest Uses a Public MD5-Derived Key (no secret, no IV)

Severity:	Medium	Probability:	Medium	Status:	Active
-----------	--------	--------------	--------	---------	--------

vault-rails encryption is gated by environment, not by whether real secrets are present: `config.enabled = Rails.env.production?` (`config/initializers/vault.rb:6`). In any non-production environment (staging, QA, CI, a developer machine), or on a production host where `RAILS_ENV` is misset, `vault_attribute` fields never reach Vault. They are handled by the fork's in-memory cipher, whose key is derived from public, non-secret metadata with no IV:

```
# plugins-vault-rails/lib/vault/rails.rb
def memory_encrypt(path, key, plaintext, ...)
  cipher = OpenSSL::Cipher::AES.new(128, :CBC)
  cipher.encrypt
  cipher.key = memory_key_for(path, key, context: context)
  DEV_PREFIX + Base64.strict_encode64(cipher.update(plaintext) + cipher.final)
end

def memory_key_for(path, key, context: nil)
  md5 = OpenSSL::Digest::MD5.new
  md5 << path
  md5 << key
  md5 << context if context
  md5.digest
end
```

For the API-key secret column the inputs are fixed and public (`path="transit", key="barong_api_keys_secret"`), so the key is recomputable by anyone. A non-production database dump (staging snapshots, CI fixtures, a developer copy of prod-shaped data) yields every API-key signing secret in cleartext, offline, with no Vault access. Because the toggle is a single environment variable, a production box booted with the wrong `RAILS_ENV` applies the same public-key obfuscation to live secrets.

Recommendations:

- Do not gate encryption on `RAILS_ENV`. Require a reachable Vault in every environment that holds real secrets, and fail closed (refuse to boot) rather than falling back to the in-memory cipher.
- Never load production-shaped data into an environment running the in-memory path; scrub or synthesize secrets in non-prod fixtures and dumps.
- Rotate API-key signing secrets if any non-production copy of production data exists or has been distributed.

Vulnerabilities Medium

M32: Fireblocks Deposit Recipient Not Pinned to the API-Verified Transaction (C1-fix residual)

Severity:	Medium	Probability:	Low	Status:	Active
Partially remediated (re-test 2026-08-04, 6539f03): the C1 handler now overrides destinationAddress/sourceAddress/txHash from the API re-fetch.					
! Outstanding to resolve: the override is conditional on the verified tx carrying destinationAddress; if absent it falls back to the webhook-body value — the recipient-substitution edge remains.					

The remediated C1 webhook (commit 6252ad5) re-fetches the transaction from Fireblocks but overrides only status, amountInfo, and feeInfo (app/api/v2/public/fireblock.rb:119-123):

```
fire_block_data = fire_block_data.merge(  
  status: verified[:status], amountInfo: verified[:amountInfo], feeInfo: verified[:feeInfo]  
) # destinationAddress / sourceAddress / txHash / blockHeight are NOT overridden  
  
# build_transaction! (fireblock_helpers.rb) then reads the recipient from the (body) attrs:  
to_address:  attrs[:destinationAddress], # <-- from the webhook body, not the re-fetch  
from_address: attrs[:sourceAddress],  
hash:       attrs[:txHash]
```

Deposit attribution credits whoever owns to_address (the M23 address match), so the amount is double-verified while the recipient is verified only by the signature. If the signature is accepted for an attacker-shaped body (see M33), an attacker references a genuine COMPLETED transaction id (passing the re-fetch) while substituting destinationAddress with their own registered deposit address, and is credited the verified amount. The idempotency layer prevents double-processing of the same tx id, not recipient substitution on a not-yet-processed one.

Recommendations:

- Take destinationAddress, sourceAddress, txHash, and blockHeight from the get_transaction_by_id response, not the webhook body.
- Assert the verified destination resolves to a known Yimmit PaymentAddress before crediting (also closes the M23 case/format gap on the trusted side).

Vulnerabilities Medium

M33: Fireblocks Webhook Signature Trust Is Config-Dependent (JWKS URL / V1 Legacy Key)

Severity:	Medium	Probability:	Low	Status:	Resolved
Remediated at 6180878 / 6ccea17 by the JWKS default was set to the production Fireblocks endpoint and a V1 legacy kill-switch added					
Resolved (re-test 2026-08-04): JWKS defaults to the production Fireblocks endpoint and the V1 legacy-key path is inert (no V1 key deployed). Residual is hardening-only (set FIREBLOCKS_DISABLE_V1_WEBHOOK=true + TLS-CA pinning).					

The verifier (`lib/fireblocks/webhook_signature_verifier.rb`) is sound in construction (RS512-only allowlist, detached-JWS over the raw body, JWKS over verified TLS, fail-closed on a missing header). Its trust depends on two configuration inputs:

- **V2 JWKS URL.** Keys are fetched from `FIREBLOCKS_JWKS_URL` (env) or the prod default. Commit 6252ad5 “correct Fireblocks JWKS URL to production endpoint” shows this was misconfigured. If it points at the Fireblocks **sandbox** JWKS (self-serve), an attacker with a sandbox account can produce a signature that verifies.
- **V1 legacy path.** The verifier also accepts a legacy `base64(RSA-SHA512(body))` signature validated by `FIREBLOCKS_WEBHOOK_PUBLIC_KEY`. If that key is a test/committed key (the C4/L24 pattern in this codebase), the V1 path is forgeable.

Impact. Chained with **M32** (recipient from the body) and a real pending tx id, a forged signature reconstitutes a credit-to-attacker path that the amount re-fetch alone does not stop.

Recommendations:

- Verify the production `FIREBLOCKS_JWKS_URL` is `https://keys.fireblocks.io/.well-known/jwks.json`; reject non-Fireblocks hosts and non-https schemes.
- Confirm the V1 legacy public key is the genuine Fireblocks key, or **disable the V1 path** now that V2 is live.
- Pin the TLS CA for the JWKS fetch; alert on JWKS-source or key-rotation anomalies.

Vulnerabilities Medium

M34: Reward-Setting *currency_id* Made Mutable Post-Creation: Operational-Role Money Lever

Severity:	Medium	Probability:	Medium	Status:	Remediated
Remediated at 8086f9e by In-place <i>currency_id</i> mutability closed at model (<i>attr_readonly</i>) and admin PUT (except).					
! Outstanding (re-test 2026-08-04): post-creation mutability confirmed locked (<i>attr_readonly</i>), but <i>currency_id</i> is still settable <i>at create</i>, and the redemption unit-confusion mint ($\sim 1/\text{price} \times \text{intended USD}$) is unchanged.					

The deployed reward changes **deleted a security invariant**. *app/models/reward_setting.rb* removed *validate :currency_id_cannot_be_changed*, *on: :update*, and the admin update path dropped its explicit exclusion:

```
# admin/rewards.rb PUT before:  r_setting.update(declared(params, ...).except(:currency_id))
# "currency not editable"
# admin/rewards.rb PUT after:   r_setting.update(declared(params, ...).to_h.except('id'))
# currency_id now writable

# trade.rb:916-923: reward minted per trade in the configured currency:
reward_currency_price = Currency.find_by(id: t_reward_setting.currency_id).price
next unless reward_currency_price.positive?
r_amount = (revenue_generated * (t_reward_setting.amount / 100.0) * fee_currency_price) / reward_currency_price
```

currency_id can now be repointed on a live setting, minting the per-trade / welcome / referral reward in that currency behind only a *.positive?* price guard. The mint-to-withdrawable chain is **confirmed**: *Redeem#perform_accounting_operation!* (*redeem.rb*) credits the reward *amount* as **USDT 1:1** regardless of its *currency_id*, yet *trade.rb* computed that *amount* in the *reward_currency*'s units. Therefore pointing the setting at a token priced *0.0001* pays out **~10,000x** the USD value as spendable, withdrawable USDT. *:manage RewardSetting* is granted to **accountant, compliance, technical, and support** (not just superadmin), and the *Redeem#compliance_role* approver gate accepts any of them.

This is a deliberate feature. The exchange-castle console has a reward-currency **Dropdown** (*RewardSettingpopup.tsx, updateRewardsSettings*), making the repoint a point-and-click action for the four roles. The real defect is the **redemption-side unit confusion** above plus the absent maker-checker.

Recommendations:

- **(primary fix)** Credit redemption in the reward's actual currency (or convert at a trusted rate) so the payout matches the computed USD value; reconcile reward *Liabilities* against payouts.
- Gate the currency change to **superadmin + maker-checker re-approval** and enforce maker-checker between reward-config and redemption-approval. (Re-add *currency_id_cannot_be_changed* only if the feature is dropped.)
- Bound *r_amount* with a per-payout cap and source *reward_currency_price* from a trusted oracle with sanity bounds (reject zero/near-zero/stale); segregate *RewardSetting* write from the broad operational roles (ties M29). [Partly fixed since audit — M29 Remediated (partial): operational roles downgraded, but *technical* retains *:write* on *RewardSetting* — residual remains.]

Vulnerabilities Medium

M35: Public JWT Signing Oracle (*/encrypt_token*)

Vulnerability severity:	Medium	Vulnerability probability:	Low	Status:	Resolved
Code-fixed <i>319d8fe</i> by The public JWT signing oracle POST <i>/encrypt_token</i> and the sibling POST <i>/decrypt_token</i> now return 410 Gone with no cryptographic operation on caller input.					

POST *.../identity/users/encrypt_token* (*users.rb:326*) is **public** and signs **arbitrary caller-supplied params** with the production private key:

```
post '/encrypt_token' do
  { token: codec.encode(params.merge(sub: 'confirmation')) }
end
```

No session-granting consumer. The minted *sub:'confirmation'* token has no endpoint that redeems it for a session or email-verification: no handler pairs *decode_and_verify(sub:'confirmation')* with *open_session* (the only other *open_session* — *confirm_otp, :419* — is OTP-code-gated), so the minted token is **currently inert**.

Recommendations:

- Remove */encrypt_token* entirely; confirmation tokens must be minted server-side from server-trusted data, never from caller params.
- Scope confirmation tokens to a single purpose and bind to a server-derived nonce; do not auto-open a session from an email-confirmation token.

M36: Cross-Service Fund Transfer Is Non-Atomic: Staking Ledger Commits Before the Peatio interchange_funds Call

Severity:	Medium	Probability:	Low	Status:	Active
Reworked 2026-07-27 (staking e381c30): the log-only after-commit call was replaced by a durable Sidekiq worker (InterchangeFundsWorker) guarded by a new interchange_confirmed_at column — closing the original commit-then-fire-and-forget non-atomicity.					
! The durable retry re-POSTs to peatio's /transfer_asset/staking, which is not server-side idempotent (no jti / idempotency-key check), so a Sidekiq retry double-moves funds. A crash between the ledger commit and the worker enqueue also strands the liability with no reconciliation sweeper.					

The staking-transaction hardening (a13b1a2 / 363756e) wraps the local ledger writes of Account#transfer_asset! (app/models/account.rb:183) in a DB transaction, then performs the cross-service fund movement to Peatio in an after_all_transactions_commit callback:

```
# app/models/account.rb (inside Account#transfer_asset!)
# ... Operations::Liability + Operations::BalanceHistory written inside the ledger transaction ...

ActiveRecord.after_all_transactions_commit do
  begin
    Management::TransferCosign.verify!(liability, member_id: member_id,
                                      currency_id: currency_id, amount: amount)

    Management::Peatio::Client.rest_api(
      :interchange_funds,
      amount: amount, type: kind, **params.slice(:uid, :currency_id, :wallet_kind)
    )
  rescue StandardError => e
    Rails.logger.error(event: "peatio_transfer_failed_post_commit",
                      member_id: member_id, currency_id: currency_id,
                      amount: amount.to_s, kind: kind, ...)

    # no compensating transaction, no retry queue, no reconciliation job
  end
end
```

The developers documented this as a deliberate tradeoff (the in-code comment notes that holding the DB transaction open across a synchronous HTTP call would block row locks and could roll back an already-succeeded remote transfer on timeout). The chosen mitigation is **log-only**, which converts an atomicity problem into a durability-of-divergence problem rather than solving it:

- **Remote failure after local commit:** if interchange_funds raises (Peatio down, timeout, 5xx, cosign rejection), the staking ledger has *already* recorded the transfer but the Peatio wallets never moved making the books and the wallets diverge in the exchange-unsafe direction, and only a log line records it.
- **Lost even the log:** the callback runs in-process after commit; if the worker/pod dies between the ledger commit and the callback executing, the Peatio call never fires and nothing is logged resulting in a fully silent divergence.
- **No self-healing:** every occurrence requires manual detection (grepping for peatio_transfer_failed_post_commit) and manual repair.

Impact. Over time this produces unreconciled drift between the staking service ledger and Peatio custody wallets — a solvency/accounting-integrity issue (funds credited/debited on one side but not the other) that is invisible to the global books until someone reads the error logs. It compounds the solvency-reconciliation gap tracked under M30.

Recommendations:

- Make the cross-service transfer a **transactional outbox / saga**: within the ledger transaction, persist a durable “pending peatio transfer” record; a separate worker delivers it to Peatio with an **idempotency key** and retries until success (or raises a compensating ledger entry).
- Drive a **reconciliation worker** that replays/alerts on any pending-or-failed transfer and a periodic sum-check between the staking ledger and Peatio balances.

- Ensure `interchange_funds` is idempotent server-side (dedupe on a transfer id) so retries after an ambiguous timeout cannot double-move funds.
- Alert on `peatio_transfer_failed_post_commit` in real time rather than relying on log inspection.

M37: Matching Executor Settles Trades Against Already-Cancelled Bot Orders

Severity:	Medium	Probability:	Low	Status:	New
------------------	--------	---------------------	-----	----------------	-----

A trading-bot stability fix ([fe28d09](#), “allow matching executor to execute trade matches on bot orders in cancel state”) relaxed the invariant that an order must be in the **wait** state to be executed, so a **bot maker/taker order in cancel state can now settle**:

```
# app/trading/matching/executor.rb
unless @maker_order.state == Order::WAIT ||
  (@maker_order.member.is_bot? && @maker_order.state == Order::CANCEL)
  raise_error(3003, "Maker order state isn't equal to wait ...")
end

if (@maker_order.state == 'cancel' && !@maker_order.member.is_bot?) ||
  (@taker_order.state == 'cancel' && !@taker_order.member.is_bot?)
  raise_error(3008, "Order already cancelled before execution ...")
end

outcome_account ... unless outcome_account.member.is_bot?
income_account ... unless income_account.member.is_bot?
```

The WAIT-state check exists to stop the engine settling against an order the operator has already pulled. The bot exception in that check ([:90](#), [:94](#)) narrows the “already cancelled” abort at [:139](#) so it no longer fires for bots while bot account mutations are skipped entirely at [:309-310](#).

Impact. A user (non-bot) order can be filled against a market-maker bot quote the platform already cancelled. It also weakens the settlement-state invariant that other engine correctness properties rely on.

Recommendations:

- Do not settle against orders in **cancel** state, bot or otherwise. Fix the root cause by making cancellation evict the order from the in-memory book atomically so a cancelled bot order can never be matched.
- If settling in-flight bot fills at cancel time is genuinely required, bound it explicitly (e.g. only for a match already in progress) and **account the bot/house side** rather than skipping it, so the books stay balanced.
- Add a regression test asserting that a match against a **cancel**-state order (bot and non-bot) is rejected or fully accounted.

Vulnerabilities Low

L1: Permissive CORS (*)

Severity:	Low	Probability:	Low	Status:	Active
-----------	-----	--------------	-----	---------	--------

`api_cors_origins = '*' (001_barong.rb:93)`. `api_cors_allow_credentials = false (:95)` and cookie-based auth block cross-origin credentialed reads. The origin-validation regex (`lib/barong/cors/validations.rb`) is loose.

Impact. `*` is broader than required; no credentialed cross-origin data exposure under current settings.

Recommendations:

- Restrict allowed origins to the known Yimmit frontends; confirm no endpoint reflects `Origin` with credentials enabled.

L2: User Enumeration via Public /status

Severity:	Low	Probability:	Low	Status:	Remediated
Remediated at <code>319d8fe</code> by Enumeration materially throttled and timing minimized via a single EXISTS query returning uniform false for missing/inactive, but a true-response existence oracle remains and the endpoint stays unauthenticated.					
! Outstanding to resolve: Residual existence oracle: <code>true</code> confirms the uid is an existing active (level>=1) account.					

`GET /identity/users/status?uid= (identity/users.rb:215)` is public and returns account status for any supplied uid.

Recommendations:

- Require auth and avoid differential responses.

L3: Missing-Authz Information Disclosure

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed <code>319d8fe</code> by Both previously-unguarded information-disclosure endpoints now enforce per-action authorization.					

`GET /admin/users/user_report (admin/users.rb:736)` returns aggregate user/KYC counts with no `admin_authorize!`; `GET /admin/abilities` leaks the caller's own permission set.

Recommendations:

- Add the missing `admin_authorize!` checks to `user_report` and `abilities`.

Vulnerabilities Low

L4: Admin Metrics Raw-SQL Interpolation

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 319d8fe by table_name/group_by validated against allowlists.					

admin/metrics.rb:14-38 interpolates from/to/action into SQL strings. Admin-gated and group_by/table_name are server-set today, so not directly attacker-reachable, but parameterize to remove the latent SQLi.

Recommendations:

- Parameterize from/to/action in admin/metrics.rb instead of interpolating them into SQL strings.

L5: Hardcoded Activation Code '12345'

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 319d8fe by send_activation_code generates a random OTP via Barong::OtpCode.					

send_activation_code (identity/users.rb:53-54) hardcodes code = '12345' (random generation commented out). The call site is currently commented (:196), so this is latent, but a backdoor-grade pattern that must not ship.

Recommendations:

- Remove the hardcoded '12345'; generate the code with SecureRandom.

L6: Global Fixed-Key Cache Write

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed c81bb3a by The KYC /callback endpoint performing the shared-key cache write was removed.					

The KYC /callback performs Rails.cache.write('params', params) (users.rb:349), writing request data to a single shared cache key; a cross-request data smear / key collision.

Recommendations:

- Remove the Rails.cache.write('params', params) shared-key write.

Vulnerabilities Low

L7: Unauthenticated Member Fee-Group Lookup

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 6180878 by Member.find_by(uid:) and the group filter were removed entirely.					

GET /api/v2/peatio/public/trading_fees?uuid= (exchange-base/app/api/v2/public/trading_fees.rb:28) does Member.find_by(uid:) and returns that member's fee group with no authentication; unauthenticated member enumeration by UID.

Recommendations:

- Require auth / drop the per-UID lookup from the public trading_fees route.

L8: Staking Committed Dev secret_key_base + Hardcoded Worker Secret

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 47505c0 / 52ec2cb by Committed dev/test secret_key_base literals replaced with ENV.					

The staking app commits dev/test secret_key_base keys in plaintext (config/secrets.yml:14,17; production uses ENV['SECRET_KEY_BASE'] at :22). A worker gates a destructive bulk update behind a committed token, ENV['STAKING_FIX_SPEED_SECRET'] == '3aa00ce71c60952b' (app/workers/speed_up_fix_worker.rb:7, marked "only for testing"), so anyone with repo access knows the literal value that unlocks it. Both persist in git history after file removal.

Impact. Repo readers obtain dev/test session-signing keys and the token unlocking a destructive bulk update.

Recommendations:

- Remove the dev/test secret_key_base and the hardcoded gate token from VCS, source them from env/secrets manager, rotate the prod session key, and remove or properly guard the test-only worker. Add a .dockerignore (staking has none) excluding .git/*.pem/*.key/secrets.yml (ties L27). The committed staking RSA management key is tracked separately as H9.

Vulnerabilities Low

L9: Admin Dashboard Outdated axios + Dead Code

Severity:	Low	Probability:	Low	Status:	Remediated
Remediated at 76254c4 by axios upgraded to 1.8.0 — closes CVE-2023-45857 and the entire pre-1.					
! Outstanding to resolve: Delete the dead crypto: the commented <code>encryptMessage</code> in <code>requestBuilder.tsx</code> and the orphaned <code>src/helpers/encryptPayload.tsx</code> (unreferenced, hard-coded RSA public key).					

The admin dashboard (`exchange-castle`) pins outdated `axios ^0.25.0 (package.json:7)`, a pre-1.0 line with known SSRF/credential-leak CVEs. Dead crypto also remains: commented-out payload encryption (`requestBuilder.tsx:40,141` references `encryptMessage()`, but the call is disabled via `const data = body;;`, sending cleartext) and an unreferenced RSA+AES `encryptPayload.tsx` helper.

Recommendations:

- Upgrade `axios` to a current release; delete the commented-out encryption code and orphaned `encryptPayload.tsx` helper.

L16: Unauthenticated Disclosure of Internal Fireblocks Asset Config

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 6180878 by the <code>expose(:options, ...)</code> block was removed, so the public currencies response no longer discloses the Fireblocks <code>asset_id</code> or gas config					

The public, unauthenticated `GET /api/v2/public/currencies` returns each network's full `options` field, including `options.asset_id` (the Fireblocks asset id) and gas config, with no auth. The `BlockchainCurrency` entity exposes `:options` unconditionally (`exchange-base app/api/v2/entities/blockchain_currency.rb:51-56`), surfaced via the public `Currency` entity. The same `asset_id` is the value handed to Fireblocks `create_transaction!` on the withdrawal path (`workers/amqp/withdraw_coin.rb:124, options.dig('asset_id')`), supplying the `assetId` used by the C1 forgery chain. Live-confirmed; the sibling `/public/networks` does *not* leak it. **Impact.** Internal Fireblocks asset ids and gas config are disclosed to anonymous callers.

Recommendations:

- Filter `:options` in the `BlockchainCurrency` entity to expose only non-sensitive fields; drop `asset_id` and gas config from public responses.

Vulnerabilities Low

L17: Anti-Phishing Code Accepted as Withdrawal Authenticator

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 8086f9e by The apc_code authenticator branch and the missing-authentication else were removed.					

The withdraw path accepts the static **anti-phishing code in place of OTP** as the second factor. `account/withdraws.rb:323-326` falls back to `params[:apc_code].eq?(current_user.anti_phishing_code)` when no TOTP is supplied. The anti-phishing code is a user-set string with no entropy, rate-limit, or rotation guarantee, materially weaker than TOTP.

Recommendations:

- Remove the anti-phishing-code fallback; require TOTP (or a short-lived, rate-limited, single-use code) on the withdraw path.

L18: Referral Share-Reward Credits Unvalidated Recipient

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 8086f9e by Implements the recommended ownership assertion: value is credited only to the caller's true referrer.					

`POST /api/v2/peatio/account/referral/reward` builds a `Reward` for `member_id: Member.find(params[:parent_id]).id` (`account/referral.rb:88-96`). It validates only existence, balance, and activation, not that `parent_id` is the caller's referrer/parent. An authenticated caller can direct reward value to any account (collusion, farming, misattribution). Rewards redeem to spendable USDT (`Redeem#perform_accounting_operation! → get_account('usdt').plus_funds, redeem.rb`), so the recipient receives real withdrawable balance, bounded by the caller's own earned-minus-redeemed balance. Feeds the M34 mint chain.

Recommendations:

- Assert `params[:parent_id] == current_user.parent_id` before crediting; reject otherwise.

Vulnerabilities Low

L19: ReDoS-Vulnerable Signup URL Validation Regex

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed c7f8074 by ReDoS regex replaced by linear-time new URL() protocol check.					

The company-signup URL-field validator ([frontend-ui Schema/signup.tsx:184](#)) uses a nested quantifier `([/\w .-]*)*` that causes catastrophic backtracking. A crafted input hangs the user's own browser (client-side self-DoS). Found by CodeQL. The pattern is not reused server-side. **Impact.** A user can hang their own browser tab; no server impact.

Recommendations:

- Collapse the nested quantifier to `([/\w .-]*)?` or use a standard URL validator; confirm the pattern is not reused server-side, where it would be a DoS.

L20: File Upload Validated by Extension Only (no MIME / magic-byte)

Severity:	Low	Probability:	Low	Status:	Remediated
Remediated at 481ca83 by Magic-byte verification remedies the extension-only weakness for signature-bearing types, but the 2GB size cap and CSV byte-check gap remain.					
! Outstanding to resolve: CSV has no magic check (HTML-as-.csv still accepted).					

The KYC/document uploader ([exchange-auth app/uploaders/upload_uploader.rb](#)) sets `extension_allowlist %w[jpg jpeg gif png pdf xls xlsx csv]` and `size_range 0...2.gigabytes` with no content-type/MIME sniffing, so a renamed HTML/SVG/script payload passes the extension check. The 2 GB cap enables storage exhaustion and DoS.

Recommendations:

- Add a `content_type_allowlist` plus magic-byte sniff, set a sane size limit, and serve stored docs with `Content-Disposition: attachment` and `X-Content-Type-Options: nosniff`.

L21: Withdrawal Destination Not Re-Validated Between Accept and Signing

Severity:	Low	Probability:	Low	Status:	Remediated
Remediated at 6180878 / 8086f9e by Auto-dispatch worker now re-validates the destination under withdraw.					
! Outstanding to resolve: Two gaps: (1) On the auto path, beneficiary-address binding is enforced only when withdraw.					

Defense-in-depth only. `workers/amqp/withdraw_coin.rb:122-128` passes `withdraw.rid` directly to Fireblocks `create_transaction!(toAddress: withdraw.rid)` at send time with no re-check that `rid` still matches the beneficiary/whitelist validated at creation. The dispatch path auto-processes unless the currency is in manual mode *and* over the per-asset limit (`withdraw.rb`). Any state-machine bug or admin mutation between accept and signing reaches the signer unchecked, compounding the C2 chain.

Recommendations:

- Re-bind and re-validate the destination (beneficiary id + on-chain address) under lock immediately before `create_transaction!`; lower and verify the auto-dispatch threshold.

Vulnerabilities Low

L22: Vault-Rails Fork Logs Sensitive Transform Value via Debug puts

Severity:	Low	Probability:	Low	Status:	Active
-----------	-----	--------------	-----	---------	--------

`transform_decode` (`plugins-vault-rails/lib/vault/rails.rb:157`) runs `puts request_opts` (`:167`) immediately before `client.transform.decode(...)` (`:168`), printing `request_opts[:value]` (set at `:160`) to stdout/logs. This is the pre-decode FPE token submitted to the Vault Transform engine, sensitive material flowing through the secrets engine. Cleartext logging of sensitive data (CWE-532); any log aggregation or temporary access exposes it. Ties M26.

Impact. Sensitive Transform values reach logs, exposed to anyone with log access.

Recommendations:

- Remove the debug `puts` (`:167`); never emit Transform values to logs.

L23: Vault-Rails Fork require "pry" in Production Crypto Library

Severity:	Low	Probability:	Low	Status:	Active
-----------	-----	--------------	-----	---------	--------

`plugins-vault-rails/lib/vault/encrypted_model.rb:1` has a top-level `require "pry"` in the production secret-handling library, loaded unconditionally via the gem entry point (`lib/vault/rails.rb:6`). `pry` is declared only as a development dependency (`vault.gemspec:25`). A prod install without the dev group raises `LoadError` at boot and breaks every model that includes `Vault::EncryptedModel`. If it loads, the dev group is shipped to prod.

Impact. Load-time boot failure across all encrypted models, or dev tooling shipped to production. No `binding.pry` was left behind.

Recommendations:

- Remove the `require "pry"`; keep `pry` in dev/test tooling only.

Vulnerabilities

Low

L24: Committed JWT Key-Pair in Repository (config/rsa-key)

Vulnerability severity:	Low	Vulnerability probability:	Low	Status:	Active
-------------------------	-----	----------------------------	-----	---------	--------

The committed key is a valid RSA private key but not the production signer, which removes the token-forgery impact (hence Low). It has not been purged, so a valid private key remains in the repository and its history. This is a secret-management failure that stays **Active** until removed. Rotate if the key is ever used for any signing role.

Live verdict: using M35's oracle we obtained a production-signed JWT and verified its RS256 signature against the committed `config/rsa-key.pub`: no match (re-confirmed 2026-06-17 with a fresh M35-oracle token). The method was validated by a positive control: the committed private key round-trips against its own public key, while the production token fails. Production signs with a distinct, properly-provisioned key. The committed key is a non-production artifact and is not exploitable.

The repository commits a valid RSA private/public key-pair at `config/rsa-key` and `config/rsa-key.pub` (fingerprint `75a19d9c...`). The Barong keystore private key is the master signing key for **three** distinct trust artifacts:

- **User bearer tokens** issued to upstream services: `Barong::Authorize#codec (lib/barong/authorize.rb:230)`.
- **Cookie/session JWTs**: `config/initializers/jwt_sessions.rb:5`.
- **Management-API JWTs** that command the Peatio wallet/ledger engine (balance and withdrawal operations): `ManagementAPIv1::Client, PRIVATE_KEY = ..keystore.private_key, RS256 (app/services/management_api_v1/client.rb)`.

The key-load order is (`config/initializers/001_barong.rb:11-34`): (1) `JWT_PRIVATE_KEY_PATH` env → (2) `credentials.yml.enc :private_key` → (3) non-production only → generate and write to `config/rsa-key` → (4) production with no key → `raise`. The committed key is a non-production artifact written by step 3. It becomes the live key only if the deployment sets `JWT_PRIVATE_KEY_PATH=config/rsa-key` or reuses this key in credentials, which cannot be determined from source.

Recommendations:

- **Remove from VCS** regardless: delete `config/rsa-key*`, add to `.gitignore`, purge from history.
- **Provision keys only via** `JWT_PRIVATE_KEY_PATH` / encrypted credentials at deploy; never let the non-prod fallback write a tracked file.
- **Segregate signing domains**: use distinct keys for session JWTs vs management-API commands so one disclosure does not grant fund-movement authority.

Vulnerabilities Low

L25: `/verify_access_token` Issues Session Cookie on Error Response

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed <code>319d8fe</code> by <code>verify_access_token</code> no longer calls <code>open_session</code> , so no session cookie is set on any response (success or error).					

The `/verify_access_token` handler (`sessions.rb:558-564`) calls `open_session(current_user, {})`, which sets the session cookie (`cookies[session_name] = ...`, `identity/utils.rb:30`) before calling `store_redis_cache(:39)`. On a forged or invalid token, `store_redis_cache` raises (the L26 `Integer > nil` bug), returning HTTP `500`, yet the response still carries `Set-Cookie: _yimmit_session=...`. Session creation is not atomic with the response: a takeover attempt (ties C2) mints a cookie on a `500`, so status-code monitoring does not see it. **Impact.** A failed token verification issues a session cookie, defeating monitoring that watches for non-200 responses.

Recommendations:

- Set the session cookie only after `store_redis_cache` succeeds, or clear it on failure.

L26: Unauthenticated 500 via `Integer > nil` in `store_redis_cache`

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed <code>319d8fe</code> by <code>store_redis_cache</code> now coerces <code>allowed_sessions</code> with.					

`store_redis_cache` (`identity/utils.rb:43-73`) evaluates `session_keys.size > options[:allowed_sessions](:66)`. The `/verify_access_token` caller passes `open_session(current_user, {})` with empty options, so `allowed_sessions` is `nil`, raising `ArgumentError` (Integer vs nil) and returning an unauthenticated `500` on a public route. Every other entry point passes `allowed_sessions: issue_session!` in `login/refresh(:339)` and `confirm_otp/confirm_code` (`users.rb:431,490`), so the `500` is unique to verify-token with no amplification. This bug currently blocks C2.

Impact. Unauthenticated 500 on the public `/verify_access_token` route.

Recommendations:

- Default/guard `allowed_sessions` (e.g. `options[:allowed_sessions] || Float::INFINITY`).

L28: Remediation Rack::Attack Throttles Reference a Non-Existent Path and Never Fire

Severity:	Low	Probability:	Low	Status:	New
-----------	-----	--------------	-----	---------	-----

The auth-hardening change (319d8fe) added two Rack::Attack throttles for the OTP endpoints (config/initializers/rack_attack.rb:46,56) keyed on identity/sessions/otp/ip and identity/sessions/resend_otp/ip. Those path fragments do not match the deployed routes, so the throttles **never match a request and never fire** — leaving the OTP resend endpoint with no application-layer IP rate limit (the exact abuse the throttles were added to stop). Any real rate limiting depends entirely on the edge (which is not in these repos).

Recommendations:

- Correct the throttle matchers to the actual request paths and add a test that asserts the throttle triggers (e.g. an N+1th request in the window returns 429).
- Do not rely on client-path string matching for security throttles; key on the resolved Grape route or controller action.

L29: Removing RequestDecryptor Also Removed Non-GET Query-String Stripping

Severity:	Low	Probability:	Low	Status:	New
-----------	-----	--------------	-----	---------	-----

The staking cleanup (70a2b20) deleted the unused RequestDecryptor middleware from config/initializers/grape.rb. As a side effect it also removed that middleware’s remove_query_params! step, which stripped the query string on every non-GET request. Query-string parameters can now merge into the request parameter hash on POST/PUT/DELETE, creating a parameter-pollution surface: a query param can shadow or supply a value the handler did not expect from the body (e.g. smuggling a field past body-oriented validation or logging).

Recommendations:

- Re-introduce explicit non-GET query-string stripping (independent of the removed decryptor), or rely on strict Grape declared(params) allowlisting on every state-changing endpoint so query params cannot leak into handler input.

L30: Captcha Fail-Open Default on the Operator-Console Login

Severity:	Low	Probability:	Low	Status:	Resolved
Captcha default changed <code>none</code> → <code>unconfigured</code> ; login now fails closed when captcha is misconfigured.					

In exchange-castle (76254c4), `captchaType` defaults to `"none"` (`src/api/config.tsx:32`, `REACT_APP_CAPTCHA_TYPE || "none"`) and the login gates its captcha logic on `captchaEnabled = captchaType() !== "none"` (`src/containers/Login/Login.tsx:214,455`). If `REACT_APP_CAPTCHA_TYPE` is unset or misconfigured at build time, the admin login renders **with no Turnstile widget and submits no `captcha_response`** — previously this value was validated unconditionally. This is a fail-open anti-bot default on the operator console. (Defense-in-depth only: Barong still validates a captcha server-side when it is configured to.)

Recommendations:

- Fail closed: default to requiring a captcha and treat a missing `REACT_APP_CAPTCHA_TYPE` as a build-time error, not a silent `"none"`.
- Keep the server-side captcha requirement enforced for the admin login regardless of the client build.

L31: New `/threshold-fee` Admin Route Omits the Client-Side Permissions Gate

Severity:	Low	Probability:	Low	Status:	Resolved
The <code>/threshold-fee</code> route now carries the permissions gate (enforcing control is server-side).					

The new ThresholdFee admin route in exchange-castle (`src/App.tsx:837-841`) is registered **without** the `permissions={userRole}` prop that its sibling sweep routes (SweepThresholds / SweepTransactions / WalletsHolding) carry, so it skips even the M14 client-side role redirect. In practice this is mitigated by the login force-logout of non-admins and by server-side authorization on `/admin/threshold_fees*`; it is a consistency gap the new feature introduced, not an independent server-side exposure.

Recommendations:

- Pass `permissions={userRole}` to the ThresholdFee route to match its siblings.
- Treat this as cosmetic hardening only — the authoritative control is server-side authz (see M14); ensure the threshold-fee endpoints enforce it.

Vulnerabilities

Informational

I1: Path-Prefix Permission Matching

Severity:	Info	Probability:	—	Status:	Remediated
Remediated at 319d8fe by Matching tightened to exact-or-path-segment boundary (@path==a.path start_with("a.path/")), eliminating substring bleed.					
! Outstanding to resolve: Blanket admin/superadmin ACCEPT api/v2 grant (denylist model) unchanged — a new sensitive endpoint with no explicit DROP is still silently in-scope.					

`validate_permissions!` matches with `@path.starts_with?(a.path)` (`authorize.rb:162`). Resolution is fail-safe (DROP overrides ACCEPT, default-deny; see the RBAC appendix), so this is not a bypass. The residual is the model it creates: admin/superadmin hold a blanket `all api/v2 accept`, making security a denylist. Every sensitive sub-path needs an explicit `DROP`, and because matching is prefix-based, a new sensitive endpoint with no DROP is silently in-scope.

Recommendations:

- Audit the DROP set on every admin-endpoint addition; prefer explicit per-resource ACCEPTs over the blanket grant. Full role×path matrix is in the Authorization/RBAC appendix.

I2: CSRF Enforcement Gated on `platform == 'web'`

Severity:	Info	Probability:	—	Status:	Resolved
Code-fixed 319d8fe by <code>platform=='web'</code> gate removed — CSRF enforced for any session on state-changing verbs.					

`validate_csrf!` (`authorize.rb:142-156`) enforces CSRF only when `session[:platform] == 'web'`, and the empty-token check at `:143` is commented out.

Recommendations:

- Confirm non-web session flows cannot reach state-changing routes without CSRF; restore the commented-out empty-token check at `:143`.

I3: Configuration Smells

Severity:	Info	Probability:	—	Status:	Remediated
Remediated at 319d8fe by removing the hardcoded 'guest' RabbitMQ password default.					
! Outstanding to resolve: the source config-defaults persist (<code>kyc_provider</code> double-assignment collapsing to <code>''</code>, <code>domain='openware.com'</code>); prod ENV overrides both (<code>BARONG_DOMAIN=https://app.yimmit.com</code>, <code>BARONG_KYC_PROVIDER=sumsub</code>), so this is code-hygiene only.					

`kyc_provider` is set twice and ends as `''` (`001_barong.rb:49/54`); defaults include `event_api_rabbitmq_password='guest'` and `domain='openware.com'`. Confirm all are overridden by production environment. No CVEs exist for the Barong/Peatio baseline; risk is concentrated in custom code and config.

Recommendations:

- Confirm `kyc_provider`, `event_api_rabbitmq_password`, and `domain` are all overridden in the production environment.

Vulnerabilities

Informational

I4: JWT Codec. Misspelled Algorithm Option and No Algorithm Allowlist

Severity:	Info	Probability:	—	Status:	Resolved
Code-fixed 319d8fe by algorithm->algorithm typo fixed; verify paths pin RS256 (decode_and_verify algorithms:"RS256"; decode_token algorithms:[@options[:algorithm]]).					

The signing codec (`lib/barong/jwt.rb`) keys its algorithm option on the misspelled `:algorithm`. Both the default and the `encode` call use the same typo, so it functions. A caller that sets the correctly-spelled `:algorithm` is silently ignored and falls through to the default. `encode` passes `@options[:algorithm]` to `::JWT.encode` with no allowlist, depending on the hardcoded `RS256` default. The verifying path (`decode_and_verify`) pins `algorithms: 'RS256'`.

Impact. Hardening. No exploitable algorithm confusion today.

Recommendations:

- Correct the `:algorithm` spelling; pin an explicit RS256 allowlist on both encode and verify; add a test that a non-RS256 token is rejected.

I5: Engine Auth Middleware: Fail-Closed Improvement With a Reintroduced Prefix Pattern

Severity:	Info	Probability:	—	Status:	Resolved
Code-fixed 6180878 by Swagger moved to PUBLIC_PATH_EXACT (exact/trailing-slash match).					

The deployed change to `app/api/v2/auth/middleware.rb` (commit range to 6252ad5) is a security improvement. A request with no `Authorization` header previously `returned` from the middleware (auth skipped, deferred downstream); it now `throws` a `401 jwt.missing` for any non-public v2 path, giving a fail-closed default-deny at the gateway-to-engine boundary.

Two issues remain. (1) The public allowlist uses `path.start_with?(prefix)`, the same prefix-matching flagged in **I1**; entries without a trailing slash (e.g. `/api/v2/swagger`) match more than intended. (2) `/api/v2/management/` is added to the bypass list, delegating its protection to the management plane's own JWT-multisig auth (see **H4**).

Recommendations:

- Match the allowlist on exact, trailing-slash-terminated prefixes (or a route allowlist). Keep the management-plane bypass under explicit review with **H4**.

Vulnerabilities

Informational

I6: Incomplete / Dead Payment-Gateway Feature. Unnecessary Attack Surface (delete it)

Severity:	Info	Probability:	—	Status:	Resolved
Code-fixed 319d8fe / 2e9fe1f by The dead payment-gateway attack surface was neutralized end-to-end: /encrypt_token (M35's root cause) and /decrypt_token now return 410 Gone with no logic (319d8fe), the zfl /latest/deposit_address that minted payment-gateway URLs was deleted (2e9fe1f), and the request-decrypt middleware remains disabled/commented-out.					

The payment-gateway / payment-link feature is incomplete and abandoned across all three repositories, yet leaves live, internet-facing endpoints in place. exchange-castle's `src/redux/constants/paymentGateway.ts` (`DECRYPT_TOKEN`, `MAKE_PAYMENT_REQUEST`, `PAYMENT_VERIFIED...`) is imported by nothing. exchange-base's `POST /latest/deposit_address` (`account/deposits.rb:182`, hard-coded `Currency.find('zfl')`) mints `https://<URL_HOST>/payment-gateway/<token>` URLs to a page that does not exist (`www.yimmit.com/payment-gateway/...` returns 404; the `app.yimmit.com` SPA bundle has no such route). exchange-auth's public `/encrypt_token` (M35) and `/decrypt_token` are a signing oracle plus token-decoder with no consumer: `callback_url/web_hook_url` are plumbed through the token but never read (confirmed 2026-06-17: an attacker-controlled `callback_url` produced zero server-side fetches). No live threat is reachable today, but this is unnecessary attack surface on the production identity API, and it is the sole reason the M35 signing oracle exists.

Recommendations:

- Delete the dead feature wholesale: the orphaned redux constants, the `zfl /latest/deposit_address` endpoint, and the public `/encrypt_token/decrypt_token` routes — this eliminates M35's root cause in one stroke. If planned, re-introduce it behind authentication with server-side token minting only.

I7: Full Security-Audit Findings/Remediation Map Committed to the Repository (**audit-test.js**)

Severity:	Info	Probability:	—	Status:	New

A console script **audit-test.js** (**frontend-ui** repo root, lines 1-125, added in **c7f8074**) was committed at HEAD. It enumerates the security-audit findings and their remediation status — effectively an internal remediation roadmap / issue map — inside a product repository. Anyone with read access to the repo (or to any build artifact or image layer that includes the source) can read exactly which findings exist and which are still open, lowering the cost of targeting the unremediated ones.

Recommendations:

- Delete **audit-test.js** from the repository and purge it from git history.
- Keep audit findings, remediation trackers, and security notes out of product source repositories; store them only in the engagement's private reporting workspace.

Financial-Flow & Engine Findings

From the ledger / internal-transfer / staking / admin-dashboard passes over `exchange-base`, `staking`, and `exchange-castle`. **Positive observation:** the core money paths in the Peatio engine — withdrawal debit, order placement/cancellation, and trade settlement — are correctly serialized with `SELECT ... FOR UPDATE` and re-checked under lock; swap pricing is server-fetched (no client price injection); and no money-creation-from-nothing exists in any happy-path. The findings below are concurrency edges, a non-atomic internal-transfer, reconciliation gaps, and broken authorization in the staking service and admin client.

Ledger & concurrency (`exchange-base`)

H6: Deposit Credit Not Idempotent — Double-Credit

Severity:	High	Probability:	Medium	Status:	Resolved
Code-fixed <code>6180878</code> / <code>2e9fe1f</code> by <code>accept!</code> runs credit + <code>Operations::Asset</code> marker creation inside <code>with_lock</code> guarded by <code>Operations::Asset</code> .					

The deposit `accept` AASM guard (`from: submitted`) is evaluated before the row lock, and the balance credit runs in the `after` callback in its own short account transaction (`app/models/deposit.rb:80-110`). Two concurrent or duplicate `accept!` on the same deposit can both credit. This compounds C1 (no webhook idempotency) and is also reachable by ordinary Fireblocks webhook retries.

Recommendations:

- Re-check the `submitted` guard while holding the deposit `FOR UPDATE`; enforce idempotency on the existing `(currency_id, txid, txout)` key before crediting.

Financial-Flow & Engine Findings

H7: Unlocked P2P / Escrow Balance Mutations

Severity:	High	Probability:	Medium	Status:	Resolved
Code-fixed 6180878 by bid_operations and all escrow/p2p mutators wrapped in with_lock/transaction with lock!, converted to update! so the balance>=0 validation (now covering escrow_balance and p2p_balance) fires.					

amount_plus/amount_subtract/swap_amount/bid_operations (account.rb:224-281) mutate p2p_balance/escrow_balance/balance via update_attribute with **no row lock, no validation, and non-atomically across columns** (e.g. credit p2p then debit balance as two statements). Concurrent operations lose updates (double-spend) and a crash between the two writes creates/destroys value. (These balances count toward the user’s spendable total.)

These mutators are not internal-only: the engine’s management plane exposes them directly: PUT /management/accounts/balance → account.swap_amount(amount, action); PUT /management/accounts/bid_amount_operations → Account.bid_operations(...); PUT /management/accounts/clear_remaining_fee → account.amount_subtract(amount, 'escrow_balance') (app/api/v2/management/accounts.rb:60,103,118). Reachability is gated by the management-JWT plane, i.e. by management-key exposure.

Account.bid_operations (account.rb:40-53) gates the transfer on account.escrow_balance.positive?, not ≥ amount_fee. So with any positive escrow (e.g. 0.01) and an arbitrary amount (e.g. 1000), the check passes, amount_subtract(amount_fee, 'escrow_balance') drives member1’s escrow **negative** (the update_attribute path skips the ≥ 0 validation — M11), and member2.get_account(currency).amount_plus(amount, 'p2p_balance') credits member2 the full amount. One legitimate-looking call **creates spendable value across the p2p ledger from nothing**, with no atomicity between the two writes.

Recommendations:

- Route the unlocked p2p_balance/escrow_balance/balance mutators through the locked with_lock primitives in a single transaction.
- Require escrow_balance ≥ amount_fee under with_lock, perform both legs in one transaction, and back it with the M11 DB CHECK constraint.

Financial-Flow & Engine Findings

M11: No DB-Level Negative-Balance Constraint

Severity:	Medium	Probability:	Medium	Status:	Resolved
Code-fixed 6180878 / 8086f9e by Four DB CHECK constraints (balance/locked/escrow_balance/p2p_balance >=0) added and present in committed schema.					

The `balance/locked ≥ 0` model validation (`account.rb:20`) is bypassed by every mutation because they use `update_columns/update_attribute`; there is no DB CHECK constraint. The only negative-balance guard is the in-Ruby `amount > balance` check, which is sound *only* where the caller holds the account `FOR UPDATE` (true for `withdraw/order/trade`; not for the P2P path, H7).

Recommendations:

- Add a DB CHECK constraint as a backstop.

M12: State-Machine / Fund-Movement Coupling Is Caller-Dependent

Severity:	Medium	Probability:	Low	Status:	Remediated
Remediated at 2e9fe1f / 6180878 by Deposit release double-debit closed via <code>with_lock+Operations::Asset</code> guard.					
 Outstanding (re-test 2026-08-04): deposit side confirmed fixed (<code>with_lock + Operations::Asset</code> guard); the <code>withdraw done/success</code> double-debit residual persists — overlapping transitions still not collapsed and take no transition-level <code>withdraw lock</code>.					

Fund movements live in AASM `after` callbacks that open their own nested transactions; only the `withdraw-create` API path wraps the whole transition in `with_lock`. Worker-driven transitions (`success!`, `dispatch!`, `release!`) and post-accept hooks have multiple commit points, so a crash between state-write and fund-move can leave state and balance inconsistent. **Modeling the machine (see the Deposit/Withdraw State-Machine appendix) makes the concrete race visible:** the terminal `succeed` state has *two* debit events (`done`, `success`) reachable from overlapping source states, and the `withdraw row` is not `lock!`d for the transition — a worker-driven race between them double-debits (the `withdraw` analog of H6’s `deposit double-credit`).

Recommendations:

- `row.lock!` + re-check state at the top of each fund-moving transition; collapse the duplicate debit events; wrap transition + fund-move + ledger ops in one transaction. Runtime-verify with a concurrent-request test (RoE-gated).

Financial-Flow & Engine Findings

M23: Deposit Attribution Is Fallback-Free & Case-Sensitive (no reconciliation)

Severity:	Medium	Probability:	Medium	Status:	Remediated
Remediated at 8220c45 by Address case-normalization added on both match sides with a case-sensitive-chain allowlist.					
! Outstanding (re-test 2026-08-04): case-normalization confirmed on both match sides, but the “reconciliation” is alert-only (a single 5-min-deduped cache entry, no queue/retry), and there is still no destinationTag / vault-account-id attribution fallback.					

Inbound on-chain deposits are credited through exactly one path (`BlockchainService.update_or_create_deposit`), and the recipient is resolved **solely** by an exact, case-sensitive address match: `PaymentAddress.find_by(wallet: Wallet.deposit_wallets(currency_id, blockchain_key), address: to_address) → return if address.blank? (blockchain_service.rb:206-207)`, then `d.member = address.member (:226)`. There is **no vault-id / destinationTag fallback and no reconciliation worker. Normal deposits credit correctly** — verified live: a real on-chain USDC/ERC20 deposit was attributed and credited, because Fireblocks reports the address in the exact stored form. The residual finding is robustness/fund-safety: if a deposit ever arrives at an address that isn’t a registered `PaymentAddress`, or whose stored case/format differs from the reported one, it is silently dropped with no automated recovery. Reachable conditions: viewing the account/deposit page does *not* create a `PaymentAddress` (only `POST /create_address` does, `account/deposits.rb:141-160`), and the match is byte-exact (a case/normalization drift between the stored value and the Fireblocks-reported value would miss). Also *scopes C1* (the unauth mint requires an exact registered-address match — a lowercase vs checksummed mismatch is why the forged-webhook PoC returned `null` while the real deposit credited).

Recommendations:

- Add a vault-account/tag attribution fallback, normalize address case on both sides of the comparison, and add an unattributed-deposit reconciliation/alert path.

Financial-Flow & Engine Findings

H13: User-Settable *is_fake* Order Flag → Account/Ledger Divergence (fee minted)

Severity:	High	Probability:	High	Status:	Resolved
Code-fixed 6180878 by <i>is_fake</i> removed from the v2/v3 param surface and hardcoded false in <i>build_order</i> .					

is_fake is a market-making settlement flag ("wash/maker order — don't charge a fee") exposed as a **user-settable order-create param with no role/bot gate** (*api/v2|v3/market/named_params.rb:39*, optional *:is_fake*; threaded onto the order at *order_helpers.rb:20* with *member: current_user*). At settlement the executor zeroes the fee on the **balance side** (*executor.rb:266*) but skips the income credit only *unless ... is_bot?* (:269-270) — gated on *is_bot?*, **not *is_fake*** — so a non-bot user who sets *is_fake:true* still has *Account.balance* credited the full, fee-zeroed income. The double-entry ledger, however, is not *is_fake*-aware: *order_fee* (*trade.rb:807*) has no *is_fake* branch and *record_liability_credit!/record_revenues!* (:994,:1036) subtract the fee unconditionally.

Net effect: every settled *is_fake* order credits the trader's *Account.balance* the full income while *Operations::Liability/Revenue* book income—fee — the account exceeds the books by exactly the fee, a **withdrawable, repeatable surplus** (fee evasion at minimum, value-from-nothing per trade; the per-order instance of M30's solvency drift).

Exploitability. *Order.submit* matches an *is_fake* order only against existing **bot** orders at the exact price/volume (*order.rb:255+*, *Member.bot_users_ids*), so an attacker must mirror a live bot quote — but bot liquidity is observable on the public depth feed, so the condition is reproducible with effort. The ledger divergence is proven deterministically (*Account.balance - Operations::Liability == fee* for a non-bot fill, collapsing to zero under the fix — *is_fake_ledger_divergence_spec.rb*, passing); the open question is **reachability**.

Recommendations:

- Remove *is_fake* and *confirmed* from the user-declared param surface (set them only on the bot/management path, server-side); make *order_fee* and the *record_liability_credit!/record_revenues!* postings agree with the account-side fee decision so balance and books can never diverge; back it with the M30 reconciliation check.

Financial-Flow & Engine Findings

H11: Authenticated SQL Injection via `Order.market_condition`

Severity:	High	Probability:	High	Status:	Resolved
Code-fixed <code>6180878</code> / <code>080f83a</code> by Both <code>market_condition</code> methods rebuilt with <code>sanitize_sql_array</code> bind placeholders / <code>conn</code> .					

`Order.market_condition` / `Trade.market_condition` (`app/models/order.rb:821-832`) [*Note — `Trade.market_condition` is actually in `trade.rb:589` and `Order.market_condition` at `order.rb:850`; both are now `Resolved` (parameterized).*] build a SQL `WHERE` array by **raw string-interpolating request params**, with no binding or escaping: `"members.uid = '#{params[:uid]}'"`, `"members.email = '#{params[:email]}'"`, `"members.role != '#{params[:role]}'"`, `"ord_type = '#{params[:ord_type]}'"` (string — quote-breakout) and `"member_id = #{params[:member_id]}"`, `"price != #{params[:price]}"`, `"origin_volume != #{params[:origin_volume]}"` (numeric — direct). The sibling `order_market_condition (:839)` does this correctly with `conn.quote/sanitize_sql_like`; the cited method does not.

It is reachable from **authenticated member** endpoints: `GET /api/v2|v3/market/orders` (`market/orders.rb:67`) and `/market/trades` (`market/trades.rb:21`) call `market_condition(params.merge(member_id: current_user.id))`. Also reachable via `admin/trades.rb:42`.

Impact. Any authenticated user injects SQL into the orders/trades query.

Recommendations:

- Parameterize `market_condition` exactly as `order_market_condition` already does (bind placeholders / `conn.quote`), or rebuild with hash-form `.where(uid: ...)` conditions; never interpolate request params into SQL. Add a regression test that `?uid=' OR 1=1--` is neutralized.

M30: Four Inconsistent “Exclude This Account” Predicates → Solvency-Invariant Drift

Severity:	Medium	Probability:	Medium	Status:	Active

The engine answers “is this account a bot / market-maker / house account that should be treated specially in fee and ledger logic?” with **four independent predicates that do not coincide**:

- **(1)** `Member#is_bot? = role == 'botcreator'` (`member.rb:139-140`) — used to skip the income credit in the executor (H13).
- **(2)** the per-order `is_fake` flag (H13) — used to zero the fee.
- **(3)** `avoidable? = cached trader_id/maker_id` tested against an `excluded_member_list` (`liability_operations.rb:6-11`) — used to skip liability operations.
- **(4)** a hardcoded `liquidity@exchange.com` / member 8 (`trade.rb:121-122,160`; `member.rb:564`) — a special-cased house account.

Because these four sets do not coincide, the same account can be treated as “house” on one ledger path and “normal” on another — the fee is zeroed on one decision, the income credited on another, and the liability skipped on a third. The global solvency invariant `sum(Account.balance) == sum(Operations::Liability)` therefore drifts, and the drift cannot be reconciled from the books because no single source of truth defines “house account.” This is the **structural root** that H13 exploits at the per-order level; on its own it is an integrity/auditability hole that makes solvency un-provable.

Recommendations:

- Collapse to **one** authoritative predicate (e.g. a single `house_account?/excluded?` consulted everywhere fee/credit/liability decisions are made), and add a reconciliation job that recomputes `sum(Account.balance) - sum(Operations::Liability)` per currency and alerts/fails CI on any non-zero drift.

Internal transfers & swaps (exchange-base)

H10: Non-Atomic Internal (User-to-User) Transfer

Severity:	High	Probability:	Low	Status:	Resolved
Code-fixed 6180878 by Internal-transfer receiver credit and sender debit now run in a single Withdraw.					

When a withdrawal resolves to another Yimmit member (internal transfer), the **receiver credit** (`deposit.accept!`) and the **sender debit** (`success! → record_complete_operations!`) run in *separate* transactions inside an `after_commit` callback with no enclosing transaction (`app/models/withdraw.rb:108-144, 570-592, 680-709`). If the sender-debit leg fails after the receiver-credit commits (transient DB error, ops failure, process kill), the receiver keeps the funds while the sender is only *locked*, never debited — value duplication with no auto-recovery (the withdraw stalls in `deposited`), and a manual retry double-credits the receiver.

Recommendations:

- Execute both legs in a single transaction; make the internal-transfer idempotent on a stable key.

M17: Swap / Bot-Transfer Double-Entry Gap

Severity:	Medium	Probability:	Low	Status:	Active
Prior remediation (6180878/8220c45): swap fee leg balanced (Revenue+Asset); <code>perform_transfer!</code> dual-locked.					
! The previously-dead <code>record_asset_debit!</code>/<code>record_asset_credit!</code> are now <i>called</i> from <code>transfer_funds</code> (<code>swap.rb:157-159</code>), but book fictional custody (no house/inventory account, <code>reference: self</code>). They only cosmetically zero the per-currency accounting equation: the real <code>Assets≠on-chain</code> gap persists and is now hidden from <code>validate_accounting_equation</code>.					

Swaps record `Operations::Liability` (debit ask / credit bid) and `Revenue` but **no `Operations::Asset`**, and bypass `validate_accounting_equation` (enforced only on `Adjustment/Transfer`) — `app/models/swap.rb:383-413`. Per-currency `Assets ≠ Liabilities + Revenue` after each swap; the swap fee `Revenue` entry is also unbalanced, and `Account#transfer_bot_asset` (`account.rb:236-262`) is the same class. No user-facing value creation (price is server-fetched, swap execution is atomic under `with_lock`), but the internal ledger is not self-balancing — a reconciliation/audit risk.

Recommendations:

- Record matching `Asset` entries (or document the off-ledger model) and extend the equation check to swaps. (Note: internal transfers also apply a *double* fee — withdraw-side + deposit-side; net-conserved but likely unintended.)

Financial-Flow & Engine Findings

Staking engine (staking)

H8: Staking Admin Authorization Is a No-Op

Severity:	High	Probability:	Low	Status:	Resolved
Code-fixed 363756e by Member#authorize! now instantiates Abilities.					

Member#authorize! is entirely commented out (returns nil), so admin_authorize never denies and the staking /admin/* mount requires only a logged-in member (admin/mount.rb:8) — a normal user could force claimable!/early_withdraw! on any stake, run plan/duration CRUD to alter interest rates, and read all members’ balances. **Live-tested 2026-06-17 (prod) and found broken-safe (cf. C2):** a member session got HTTP 200 on /staking/stake/investments but 401 authz.invalid_permission on /staking/admin/investments — the Barong gateway is default-deny and the only member→staking rule (seeds.yml:294) does not prefix-match /admin. So it is a confirmed defense-in-depth failure but **not externally exploitable today** (prob Low). It flips live if staking is reachable off the gateway (direct ingress / SSRF) or a broad { role: member, path: api/v2/staking, accept } rule is added (the I1 prefix-match would extend it onto /admin). M35 does not unlock it (sub mismatch); **C6 supersedes it** — the committed admin build credential reaches the staking source directly.

The interest-rate edit is a fund-minting chain. Staking reward = amount × duration.interest (investment.rb:5); on claim, principal+reward moves into Peatio spot via interchange_funds → POST /transfer_asset/, **signed with the committed staking key (H9)**. So: edit a duration’s interest → stake dust → claim an inflated reward → withdraw real assets. **Effective impact is theft of funds (Critical-class)**, gated only by whether Peatio accepts staking’s single management signature (H4). Safety rests on H8 (rate integrity) → H9 (key secrecy) → H4 (signer threshold) → a server-side reward cap — all four currently weak.

Recommendations:

- Restore role enforcement in authorize! (a stubbed TODO); scope admin lookups; add the missing admin_authorize on BalanceHistory; **and independently** bound the reward amount Peatio accepts via interchange_funds and require ≥2 management signers (H4) for staking transfers, so one broken control can’t mint funds.

H9: Staking Management Signing Key Committed

Severity:	High	Probability:	Low	Status:	Active
Remediated (env): the RSA key is now read from ENV (47505c6/363756e); no key in the HEAD tree.					
 Outstanding to resolve (re-test 2026-08-04, downgraded from Resolved): the private key remains recoverable from git history and rotation is unverified from the repo. Purge history + rotate.					

conditional · staking/config/staking.yml:4 commits the RSA private key used by Management::Base#generate_jwt to sign interchange_funds/get_balance management calls to Peatio. If production uses this committed key, anyone with repo access forges signed management JWTs and moves funds between any user’s spot/staking wallets.

Recommendations:

- Verify prod does not use it (same test approach as L24, which was disproven) and purge from VCS; confirm Peatio requires ≥2 signers for these scopes (H4).

Financial-Flow & Engine Findings

M13: Staking Double-Claim Race on Unstake

Severity:	Medium	Probability:	Low	Status:	Resolved
-----------	--------	--------------	-----	---------	----------

Code-fixed `a13b1a2` / `363756e` by Unstake PUT wraps the state check and transition in a DB transaction with a pessimistic `@investment`.

PUT `/stake/investments` reads investment state and runs guards with no row lock before the transition (`investments.rb:96-115`); the flexible `claimable` guard is count-based, not state-based. Concurrent claims on a matured investment may run the credit path twice.

Recommendations:

- `current_user.investments.lock.find` before the checks. Needs runtime verification of AASM re-evaluation.

Admin dashboard (exchange-castle)

M14: Authorization Is Client-Side Only

Severity:	Medium	Probability:	Medium	Status:	Resolved
Remediated at af9737b / 3cfc093 by disableCheck corrected (disables for any role ≠ admin/superadmin), a permissions=userRole prop threaded through PrivateRoute with a login-redirect, and non-admins are force-logged-out at login.					
Resolved (re-test 2026-08-04): server-side per-action authorization is now enforced (see H5), so the client-side dashboard gate is no longer the security boundary; residual is cosmetic (UX).					

The console is gated on state==='active' with **no role check** in PrivateRoute (selectors/user.tsx, routes/index.tsx), and privileged buttons use a decorative, inverted disableCheck (disables only for role==='admin', enables for everything else) — trivially bypassed by dispatching the saga directly. Every privileged action (withdrawal/deposit approval, balance adjustment, role change, KYC approval, premium toggle) relies entirely on server-side admin_authorize!. The dashboard wires up 3 of the 4 H5-unprotected endpoints (update_premium, users/comments, admin/profiles), substantiating H5’s impact.

Recommendations:

- Server-side authorization is the real control (see H5); also enforce role in the route guard and fix disableCheck.

M15: CSRF Token in JS-Readable Cookie

Severity:	Medium	Probability:	Low	Status:	Active
-----------	--------	--------------	-----	---------	--------

The admin CSRF token is stored in a non-HttpOnly cookie read by JavaScript (helpers/getCsrfToken.tsx) to set X-CSRF-Token. Any reflected/stored XSS or malicious dependency can read it and forge state-changing admin requests, defeating CSRF protection (no XSS sink was found in this app, so impact is conditional).

Recommendations:

- Double-submit via an HttpOnly + readable split, or bind CSRF to the HttpOnly session server-side.

Financial-Flow & Engine Findings

M16: Operator Console Missing Security Headers

Severity:	Medium	Probability:	Medium	Status:	Resolved
Code-fixed <code>c7f8074</code> / <code>76254c4</code> by Both the user SPA (frontend-ui) and the finding's actual target — the exchange-castle operator console — went from zero security headers to a full set with <code>always</code> : HSTS, X-Frame-Options DENY, nosniff, Referrer-Policy, Permissions-Policy, and CSP <code>`frame-ancestors 'none'`</code> .					

`exchange-castle/nginx.conf` sets no `Content-Security-Policy`, `X-Frame-Options/frame-ancestors`, `HSTS`, `X-Content-Type-Options`, or `Referrer-Policy` means clickjacking exposure and weakened XSS containment on a privileged console.

Recommendations:

- Add a strict CSP and the standard header set to the operator-console nginx, and confirm the SPA HTML document response (not just the API) carries `X-Frame-Options/CSP frame-ancestors`.

Authentication — login flow (exchange-auth/sessions.rb)

M18: User Enumeration on Login (Response & Timing)

Severity:	Medium	Probability:	High	Status:	Resolved
Remediated at 319d8fe / 694b567 by unifying the login response body to 'Invalid credentials' for wrong-password vs unknown-user.					
All cases converge on invalid_login! → 401 (unified body + code), and a fixed-cost dummy-bcrypt runs for unknown users.					

Login resolves the user first and errors out for a non-existent account *before* running bcrypt (`get_user` → `error!`, `sessions.rb:14-30`), so existing accounts incur the bcrypt delay and non-existent ones do not — a **timing oracle**. Responses also differ by state: non-existent (`invalid_params` 401), banned (`'account is banned'`), deleted (`'account is deleted'`), inactive, vs wrong-password (`invalid_params` 422); `/jwt` returns an explicit `'user_doesnt_exist'` (:514); the email-verify endpoints (`users.rb:464,497,519`) and registration similarly differentiate (password-reset, by contrast, correctly stays silent). **This directly answers the password-hashing-stability question: existing vs non-existing accounts are NOT processed uniformly.**

Recommendations:

- Always perform a bcrypt comparison against a fixed dummy hash for unknown users, and return one generic error + status for all credential failures.

M19: 2FA Enable/Disable Not Gated on Password

Severity:	Medium	Probability:	Medium	Status:	Resolved
Code-fixed 319d8fe by Both endpoints now require the account password (authenticate) in addition to a valid, non-replayable TOTP.					

TOTP `enable/disable` (`resource/otp.rb:53,89`) require an authenticated session + a valid TOTP code, but **not** the account password. An attacker on a hijacked session (e.g. via C2) can enroll a fresh TOTP on the victim or remove an existing one.

Recommendations:

- Require password (or step-up) for 2FA changes. [Fixed since audit — **Resolved**: 2FA disable now requires the acting admin's TOTP (M24, 319d8fe), and self-service 2FA enable/disable now requires the account password (M19).] (Also: TOTP enrollment is otherwise well-gated — no Sakurity-class ungated enable, no recovery-without-2FA path.)

Financial-Flow & Engine Findings

L10: Password-Reset Missing Rate-Limit / Resend Throttle

Severity:	Low	Probability:	Low	Status:	Resolved
Remediated at 319d8fe by adding a working Rack::Attack throttle on the password-reset endpoint (rack_attack.rb:50 — 10 / 5 min per trusted-proxy IP).					
A per-email throttle (3 / 15 min, parsed from the request body so rotating IPs cannot bypass) now supplements the per-IP throttle.					

Password-reset (users.rb:618) has no per-account/IP rate-limit or resend throttle (CAPTCHA-only) → reset-email spam / token-grinding. [Correction — now false: rack_attack.rb:50 throttles the reset endpoint (identity/users/password/generate_code) at 10 / 5 min per trusted-proxy IP. Per-IP only (a distributed sender is slowed, not blocked) and no per-account cap.]

Recommendations:

- Add per-account and per-IP rate-limiting plus a resend throttle to password-reset (beyond CAPTCHA).

L11: TOTP Reuse Check Commented Out

Severity:	Low	Probability:	Low	Status:	Resolved
Code-fixed 319d8fe by TOTP reuse check re-enabled; a code cannot be replayed within its skew window.					

The TOTP reuse check is commented out (totp_service.rb:46), so only validity (not single-use) is enforced → a code is replayable within its skew window.

Recommendations:

- Re-enable the TOTP single-use / reuse check so a code cannot be replayed within its skew window.

Financial-Flow & Engine Findings

L12: EncryptionService Key Fallback to 'Test' + /refresh DoS

Severity:	Low	Probability:	Low	Status:	Remediated
Remediated at 319d8fe by Key fallback to 'Test' removed (now raises), but the /refresh nil-cache NoMethodError DoS is unfixed.					
! Outstanding to resolve: /refresh NoMethodError on a missing/expired cache key remains unguarded (sessions.rb:516-517).					

EncryptionService derives its key from ENV.fetch('SECRET_KEY_BASE','Test') (encryption_service.rb:56) — falls back to the constant 'Test' if unset (outer layer only; inner JWT still signed); and /refresh raises NoMethodError on a missing cache key (sessions.rb:707, minor DoS).

Recommendations:

- Remove the 'Test' key fallback (require SECRET_KEY_BASE); guard the missing-cache-key path in /refresh against NoMethodError.

User frontend (frontend-ui)

L13: Frontend Residual Hardening

Severity:	Low	Probability:	Low	Status:	Remediated
Remediated at 3c75c62 / c7f8074 by Two of the four subitems now addressed (not one as stage-1 stated): reset-token capture-and-scrub (3c75c62) AND tabnabbing rel=noopener added to all live external links.					
! Outstanding to resolve: axios still ^0.27.2 (pre-1.0 CVE line, NOT upgraded).					

Outdated axios + npm audit findings (both frontends; upgrade axios specifically for the SSRF/credential-leak CVE line); uncoded backend errors can pass through i18n to the UI (goAlert.tsx:81 — depends on the backend always returning coded errors); password-reset token still read from the URL query (ChangePassword.tsx:42); ~18 external target="_blank" without rel="noopener" (tabnabbing).

Recommendations:

- Upgrade dependencies, add a generic fallback for unmapped error strings, move the reset token to fragment/POST, and add rel="noopener" to external links.

Financial-Flow & Engine Findings

Realtime / WebSocket (peatio-core ranger)

M20: Ranger Edge-Authorization Gaps

Severity:	Medium	Probability:	Low	Status:	Active
-----------	--------	--------------	-----	---------	--------

The ranger WebSocket server ([peatio-core/lib/peatio/ranger/web_socket.rb:24](#), [connection.rb:98-115](#)) authenticates via a gateway-injected **Authorization** JWT but (a) performs **no Origin allow-list** (none visible at the Ambassador gateway either) — a cross-site WebSocket hijacking risk if edge auth relies on ambient cookies; (b) trusts the injected header with **no secondary session binding**, so anything that reaches [ws.local:8081](#) directly (bypassing the gateway) can present any valid JWT; and (c) the [/private](#) vs [/public](#) path split is a frontend convention, **unenforced server-side**. Both depend on gateway/network-segmentation config not in the repos. **Note:** ranger’s private auth is the *same Barong JWT* as the REST API, so C2/M1-class forgery would equally grant a victim’s private WS feed.

Recommendations:

- Enforce an Origin allow-list and verify the WS handshake against the session at the edge; ensure ranger is not directly reachable.

L14: Ranger Plaintext WS + PII in Debug Logs

Severity:	Low	Probability:	Low	Status:	Active
-----------	-----	--------------	-----	---------	--------

Ranger terminates plaintext WS ([secure: false](#)) relying on gateway TLS, and writes full private message payloads (balances/orders/withdrawals) to debug logs ([connection.rb:25-35](#)).

Recommendations:

- Ensure TLS to the edge and an unsniffable internal network; redact private payloads from logs.

Financial-Flow & Engine Findings

Infrastructure & configuration

H12: Committed OKX Exchange API Credentials (key + secret + passphrase)

Severity:	High	Probability:	Medium	Status:	Remediated
Remediated at <code>df9c2b6</code> / <code>8220c45</code> by OKX client/configuration files carrying the hardcoded credential fallbacks deleted from the working tree (<code>df9c2b6</code>) and OKX/borrow platform-settings columns dropped by migration (<code>8220c45</code>).					
! Outstanding to resolve: Rotate the OKX credentials exchange-side and scrub git history / old image layers — prod runs a <code>changeme</code> placeholder, so the committed key is not the live one.					

A complete OKX trading-API credential set is committed in `exchange-base` as hardcoded `|| 'literal'` fallbacks behind the ENV lookups, in two places:

- `lib/peatio/okx/api/configuration.rb:17,22,27`.
- `lib/peatio/upstream/okx/client.rb:35-37`.

These are not placeholders: they sign live OKX requests such as `OK-ACCESS-KEY/OK-ACCESS-PASSPHRASE` headers and an HMAC-SHA256 over the request with `secret_key` (`okx/api/request.rb:86-107`, `upstream/okx.rb:227`). The `||` pattern means an unset ENV silently falls back to the committed key, and either way the secret is burned in git history (and baked into images via L27).

Impact. Anyone with repo or image access holds the OKX account’s full API credentials. Depending on the key’s OKX scopes (read / trade / withdraw), that is unauthorized trading or fund movement on the exchange’s OKX liquidity/upstream account. Treat as compromised.

Recommendations:

- Rotate the OKX key/secret/passphrase now.** Remove the hardcoded fallbacks and require `OKX_API_*/UPSTREAM_OKX_*` from a secrets manager and fail closed if unset; purge from history; fold into the org-wide committed-secret rotation (ties C3/C4) and image hygiene (L27).

M27: Vulnerable & Outdated Dependencies (gems + npm)

Severity:	Medium	Probability:	Medium	Status:	Active
-----------	--------	--------------	--------	---------	--------

OWASP A06 (Vulnerable & Outdated Components). Dependency hygiene is unenforced across the repos — lockfiles pin old, known-vulnerable versions with no automated audit gate. The confirmed instance is `axios ^0.25.0` (a pre-1.0 line with published SSRF / credential-leak CVEs) in both `exchange-castle` and `frontend-ui` (ties L9, L13); the Ruby side ships unaudited gem versions across `exchange-base/exchange-auth/staking`.

Impact. Known CVEs in shipped dependencies are exploitable without any custom-code bug; the `axios` SSRF/credential-leak line is reachable from the admin and user frontends. *[Specifics:the axios issue is CVE-2023-45857 (SSRF / credential leak, fixed in ≥1.6.0). At HEAD, exchange-castle upgraded axios to ^1.8.0, but frontend-ui is still on ^0.27.2. There is still no CI audit gate (no Dependabot/renovate/npm-audit) in any repo.]*

Recommendations:

- Add `bundler-audit` and `npm audit` (or Dependabot) as a CI gate; upgrade flagged dependencies starting with `axios` to a current 1.x; pin and refresh lockfiles. ECR scan-on-push covers image OS packages, not these app-level gem/npm CVEs.

Financial-Flow & Engine Findings

L27: No .dockerignore — .git/Secrets Baked into Image Layers

Severity:	Low	Probability:	Medium	Status:	Resolved
Code-fixed 73ace3f / 6180878 by A .dockerignore was added to all three delta Rails repos, each excluding `.					

The Rails repos (exchange-base, staking, exchange-auth, peatio-core, plugins) ship **no** .dockerignore and build with COPY . \$APP_HOME (exchange-base/Dockerfile:41), so the whole build context — including .git/ and config/secrets.yml — is baked into image layers. Because .git/ is copied wholesale, keys removed from the working tree still ride into every built image via history: exchange-base’s config/private.pem and Partner_test_private_key.pem were deleted in e4598f7 (2026-04-29, before the reviewed commit) yet remain reachable in the packed .git of any image built from this context. A working-tree cleanup alone does not evict them from already-built layers.

Recommendations:

- Add a .dockerignore to every Rails repo excluding .git/*.*.pem/*.*.key/secrets.yml; rebuild so .git history is no longer baked in; and rotate any key that was ever committed, since it persists in published layers regardless of working-tree deletion.

M21: force_ssl Fail-Open Default (engine + staking)

Severity:	Medium	Probability:	Low	Status:	Resolved
Code-fixed 6180878 / 47505c0 by Both engine and staking default force_ssl to true via ENV.					

exchange-base and staking production set config.force_ssl = ENV['FORCE_SECURE_CONNECTION'] == 'true' (production.rb:57) — off unless the env var is set → no HSTS, no HTTP→HTTPS redirect, and session/auth cookies not flagged Secure. exchange-auth correctly hardcodes force_ssl = true.

Recommendations:

- Default to enforced TLS; verify the env is set in EKS.

Financial-Flow & Engine Findings

L15: Infra Config Hygiene (Low)

Severity:	Low	Probability:	Low	Status:	Remediated
Remediated at 319d8fe / 8086f9e by Multiple sub-items fixed: MaxMind key ARG scrubbed to REDACTED placeholder (319d8fe).					
! Outstanding to resolve: Not fully closed: (a) plugin base images still FROM rubykube/peatio:latest in exchange-base/Dockerfile.					

MaxMind license key as a Dockerfile ARG default (exchange-auth/Dockerfile:18); VAULT_TOKEN:changeme / VAULT_DEV_ROOT_TOKEN_ID:changeme weak defaults (prod fails closed); unpinned FROM rubykube/peatio:latest plugin base images (non-reproducible); exchange-base/config/secrets.yml committed dev/test key (new instance of L8 class; prod env-sourced); castle nginx missing server_tokens off (frontend-ui has it); SPA try_files appends raw \$request_uri + no client_max_body_size; a second Barong Rack::Cors at middleware position 0 over origins '*' / resource '*' (credentials off, so not an exfil vector but broader than L1).

Recommendations:

- Move the MaxMind key out of the Dockerfile ARG, pin plugin base images, remove the committed dev/test secrets.yml, set server_tokens off on castle nginx, add client_max_body_size, and scope the second Rack::Cors.

Provenance, Blast Radius & Integrator Trust

Yimmit is a fork of the open-source OpenDAX stack ([openware/barong](#) for auth, [openware/peatio](#) for the engine; [VERSION 3.2.0](#)). Almost every Critical/High lives in custom code (the bespoke Fireblocks integration, the custom auth/identity endpoints, custom features (swap, P2P, internal transfer), and committed config/secrets), not in the OpenDAX core. Where the upstream primitives are used unmodified (the account-locking model, the deposit/withdraw state machines), they are largely sound (see the positive ledger-atomicity observations).

Per-finding attribution (representative)

Finding	Layer	Evidence
C1 Fireblocks webhook (mint)	Custom	Fireblocks integration absent from upstream Peatio; block-scanner replaced by webhook
C2 <code>/verify_access_token</code> (unverified decode)	Custom	Endpoint not in upstream Barong; opens a session from an unverified token
M35 <code>/encrypt_token</code> signing oracle	Custom	Custom withdrawal/confirmation-token endpoint
C3/C4/C6 committed secrets	Custom	Jenkins client, Gemfile PAT, Event-API key — operator config
M1 bearer <code>.decode</code> (verify=false)	Custom	Added on the urkishan/peatio-core feature/added-decode-method fork branch
H7 P2P/escrow unlocked mutations	Custom	<code>p2p_balance/escrow_balance</code> columns + methods are custom additions
H10 / M17 internal transfer & swap	Custom	Internal-transfer and Binance/OKX swap are custom features
H5 missing <code>admin_authorize!</code>	Custom	On custom admin endpoints (<code>update_premium</code> etc.)
H1 gateway XFF trust	Barong-derived (modified)	<code>AuthorizeController</code> is upstream; XFF handling diverges
Core ledger locking (withdraw/order/trade)	Upstream (sound)	Row-locked, atomic — the positive observations

Provenance, Blast Radius & Integrator Trust

Attribution: confirmed by upstream diff

The tags above were verified by a file-level diff against freshly-cloned upstream baselines: **openware/peatio@bafe5303 (v2.6.0)** and **openware/barong@1f488179 (v2.6.0)**.

- **Peatio:** `public/fireblock.rb`, `fireblock_helpers.rb`, `lib/fireblock/`, and all `fireblock/vault_id/Fireblock::Account` references — **0 hits upstream** (C1 custom). Upstream instead ships a block-scanner daemon (`app/workers/daemons/blockchain.rb`, **present**) — confirming the trust-boundary swap. `p2p_balance/escrow_balance` and the unlocked mutators — **absent upstream** (H7 custom); the locked, double-entry `plus/sub/lock/unlock_funds` path is **preserved from upstream** (the sound core).
- **Barong:** `/verify_access_token`, `/encrypt_token`, `/decrypt_token` — **0 hits upstream** (C2/M35 custom). The `decode_token` method (`JWT.decode(..., false)`, verification OFF) is **fork-added**; every upstream `JWT.decode` uses `true` (M1/H1 custom).

*Caveat: the forks self-report **VERSION 3.2.0**, which is not a real openware tag (a Yimmit-internal bump); the nearest genuine upstream is the **2-6-stable** line used above. The **Custom** attributions are robust to this (entire files/methods are flat-out absent upstream); the **Modified** ones (H7, H10) are diffed vs 2.6.x and may carry minor upstream-evolution noise. Net conclusion stands: **the Critical/High findings are introduced in the custom layer, while the OpenDAX core (ledger locking, double-entry, deposit/withdraw state machines) is preserved and sound.***

Appendix: Positive Observations

Items where the platform is doing the right thing — called out so they remain protected through future changes. Several are *tested-and-found-sound* results from this engagement: a finding that did **not** materialize is itself a deliverable, and the contrast sharpens where the real risk concentrates (the custom integration layer, not the OpenDAX core).

P1 — The OpenDAX/Peatio ledger core is preserved and sound. The money paths in the engine — withdrawal debit, order placement/cancellation, and trade settlement — are serialized with `SELECT ... FOR UPDATE` and re-checked under lock, use double-entry `Operations::{Asset, Liability, Revenue}`, and contain **no money-from-nothing path in any happy flow**. The upstream-diff (vs `openware/peatio@bafe5303`) confirms the locked `plus/sub/lock/unlock_funds` primitives are unchanged from upstream. The Critical/High findings live in the *custom* layer bolted on top, not in this core.

P2 — The gateway→engine bearer is cryptographically verified. Peatio verifies the forwarded Barong-minted `Authorization: Bearer` against the JWT **public key** (`peatio-core decode_and_verify_token` → `JWT.decode(..., true, verify_options)` with `verify_sub/exp/jti/iat`). Header-injecting a forged bearer directly into the engine therefore fails the signature check — the inter-service trust core is sound, and this is what *bounds* the blast radius of the auth-service oracles (M35/C2) to the Barong session plane.

P3 — Every inbound webhook except Fireblocks is authenticated. The Transak callback verifies a per-record HS256 JWT (`public/transak_callback.rb`) and the Sumsuub KYC webhook verifies an `X-Payload-Digest` HMAC (`public/general.rb`). The platform already owns the correct pattern in-house — which is precisely why C1 (the one money-moving callback that verifies nothing) is an outlier to fix, not a missing capability.

P4 — Swap pricing is server-fetched, not client-supplied. Swap/exchange prices are taken from the server / upstream venue (OKX `fillPx/accFillSz`), not from request parameters, so a client cannot inject a favorable price. *[Update — the OKX attribution is stale: OKX was removed (H12), and `Swap#fetch_price (swap.rb:106)` now prices via **Binance** (`api.binance.com/api/v3/ticker/price`). The positive still holds (server-fetched, not client-supplied); but note the fallback at `swap.rb:114` is a **silent rescue 0** — a price-fetch failure yields a zero price rather than aborting the swap, which should be bounded/raised, not swallowed.]* (The residual M17 concern is double-entry completeness on the swap legs, not price injection.)

P5 — Real on-chain deposits attribute and credit correctly. Verified live: a genuine ERC-20/USDC on-chain deposit was attributed to the correct member and credited through the normal path. The deposit-attribution finding (M23) is a robustness/fallback gap for edge cases, not a break in the happy path.

P6 — The password-reset flow is robust. Reset uses a scoped `sub:'reset'` token through `EncryptionService`, consumed single-use — structurally separate from, and unaffected by, the `encrypt_token/decode_token` codec misuse (M35/C2).

P7 — Realtime (ranger) channels are uid-scoped; no cross-account subscription. A user cannot subscribe to another account's private stream. M20 is edge-auth / Origin-allowlist hardening (CSWSH, header-trust), **not** a cross-account data leak.

P8 — The user frontend's prior-audit Criticals are remediated. All four prior Criticals are fixed (CSRF token moved off `localStorage` to a `Secure;SameSite=Strict` cookie; no `window.env` config exposure; `window.open`/iframe origin-validated; KYC iframe + `postMessage` pinned to `app.shuftipro.com`). Blockchain address validation is **fail-closed** (`multicoin-address-validator`) at both the beneficiary schema and the withdraw form; source maps are disabled; no hardcoded secrets in frontend source.

P9 — The spoofable-XFF pattern does not replicate into the engine or staking. The leftmost-`X-Forwarded-For` / `True-Client-IP` trust (H1) is confined to the Barong gateway; the Peatio engine and staking service do not repeat it.

P10 — No host-header / email-link poisoning. Password-reset and email-confirmation links are built from server-side config (`Barong::App.config.domain`), not the request `Host/X-Forwarded-Host` header — so reset/confirm links cannot be poisoned to an attacker domain. (Tested this engagement.)

Appendix: Positive Observations

P11 — No client-controllable authorization-path header. The gateway authorizes the routed URL glob (`params[:path]` from `match '/api/v2/auth/*path'`), not an `X-Original-URL/X-Rewrite-URL` header — so the naive path-override smuggling vector against the `starts_with?` rules is closed. (Tested this engagement; the residual normalization-divergence concern is deployment-dependent and noted under I1.)

P12 — The committed JWT key-pair is not the production signer. L24, live-tested: a production-signed token does **not** verify against the committed `config/rsa-key.pub`. Production signs with a distinct, properly-provisioned key — the committed artifact is non-production (repository-hygiene only, not a key disclosure).

P13 — The deployed branch is more hardened than the audited snapshot. The audited working tree (the initial-commit snapshot) predated the deployed code: several issues that existed *only* in that snapshot had already been remediated on `origin/production` before the engagement began — a hardcoded `'12345'` login OTP (now random generation), an `antiersolutions` backdoor email, the Fireblocks webhook SQL injection (now a parameterized bind), and the Jumio KYC callback (replaced by an HMAC-verified Sumsb webhook). These are **not carried as findings** — they were never live on production during the engagement — but they confirm the snapshot was stale relative to the deployed code.

P14 — The edge layer adds defensive headers. The live edge (Envoy/CloudFront) returns `X-Frame-Options/CSP/nosniff/Referrer-Policy` and `HSTS` on observed API responses, and `HSTS` was observed on the Peatio response — reducing the real-world exposure of M16/M21 (which stand as defense-in-depth on the origin/repo config). Containers run as non-root and no Rails `master.key` is committed.

P15 — JWT private-key loading is fail-closed in production. The key-load order (`config/initializers/001_barong.rb:11-33`) is: `JWT_PRIVATE_KEY_PATH` file → Rails encrypted credentials → (non-production only) generate-and-save → **else raise 'Private key not found or invalid'**. The committed `config/rsa-key` is only ever *written* by the dev/test generate branch; it is **never read as a production fallback**. A production deploy with no configured key **refuses to boot** rather than silently signing with a weak/known key. This is the correct design and corroborates the L24 live result (the committed key is not the production signer). Residual hygiene: remove `config/rsa-key/.pub` from the image so `JWT_PRIVATE_KEY_PATH` cannot accidentally point at it (ties L27).

P16 — Management-API multisig is server-enforced and fail-closed at config-load. The management middleware verifies signatures with `JWT::Multisig.verify_jwt` against a keychain of **public** keys and enforces the threshold as `mandatory_signers ⊆ verified` (all required signers must sign, server-side — `jwt_authentication_middleware.rb`). The config loader (`config/initializers/jwt.rb`) **raises at boot** if any keychain entry is a private key, or if any scope's `mandatory_signers` is empty; an empty keychain bricks the management API (401s) rather than opening it. The mechanism is correct — the residual (H4) is purely the *deployed* threshold count, which is operator-supplied and absent from the repos.

P17 — Auth0 and password-reset token verification are correct. `Barong::Auth0::JWT.verify` checks signature + `iss` + `aud` with `RS256` pinned, resolving the key from the Auth0 JWKS by `kid` (no algorithm confusion) — and the `/auth0` login endpoint is **disabled** (commented out) on production regardless. Password-reset tokens are scoped (`sub:'reset'`), single-use and expiring, and are not forgeable via the M35 oracle. The verifying codec path (`decode_and_verify`) pins `algorithms: 'RS256'` and verifies `iss/aud/exp/jti/sub` — the unsafe path is exclusively `decode_token` (C2).

P18 — API-key authorization is path-scoped and default-deny. API keys are restricted to the `api/v3/liquidity/` and `api/v3/ranger/` surfaces only (`validate_api_key_permissions!`, `authorize.rb:291`) — they cannot reach the general member API. Capability gating is fail-closed: `GET` requires `can_read`; non-`GET` is permitted only for an explicit withdraw path (`can_withdraw`) or trading path (`can_spot_trade`), and **any other path returns false (denied)**. An unknown state-changing endpoint is rejected, not over-granted. (The separate weakness is that the IP-allowlist binding keys off the spoofable `remote_ip` — H1.)

P19 — KYC tier promotion is gated on authoritative Sumsb data. A user cannot self-assert a verified tier: `tier_label_update` sets the verified label **only** when `SumsbKycService#applicant_status` reads `reviewAnswer == 'green'` from a live Sumsb API call against the user's own applicant (`sumsub_kyc_service.rb`), and the verify trigger is the HMAC-verified webhook (P3). The promotion *source* is sound; the residual risk is the disabled identity-correlation control (`registered_number_match_with_sumsub?` commented out — M22) and the caller-supplied `applicant_id` binding, both bounded by the `SumsbApplicant` PK constraint.

Appendix: Positive Observations

P20 — Admin mutations carry partial guard-rails. The admin user-update path blocks enabling 2FA on another account, blocks self-modification, gates email-change to `superadmin`, protects superadmin targets from non-superadmins, and requires the admin's own `otp_code` for `delete_user`. (The gap is that the same step-up is *not* required to **disable** a user's 2FA — M24.) *[Fixed since audit — Resolved: `admin/users.rb:161-165/:343-347` now require the acting admin's own TOTP (`otp_code`) to disable a user's 2FA (319d8fe); this step-up gap is closed.]*

P21 — Profile and account-state mutations are self-scoped and allowlisted. Profile edits resolve the target as `current_user.drafted_profile` (no caller-supplied id — no BOLA) and write only an explicit `declared(params.slice(*profile_param_keys))` allowlist, gated on editable KYC state (`resource/profiles.rb`). Account-state changes (deactivate/delete/password) all act on `current_user` only. The one residual is the validation-skipping `update_columns` on the preference route — bounded to three enum-validated fields, no privilege escalation (M7). (Smell to clean up: a commented-out `current_user.update(level: 3)` self-promotion line at `resource/users.rb:325`.)

P22 — Referral reward reads are self-scoped; redemption is balance-bounded. All referral/reward listing and totals query `current_user.rewards` (no caller-supplied member id — no BOLA), and redemption is capped at the caller's own earned-minus-redeemed balance (`account/referral.rb:48-102`) — no over-redeem or value-from-nothing. (The exception is the share-reward recipient, which is unvalidated — L18.)

P23 — No SSRF, SSTI, command-execution or unsafe-deserialization sinks found. Outbound HTTP targets are fixed/config hosts only (Jumio `ENV`, captcha provider, a constant `api.blockcypher.com`, the management base URL, the payment-gateway) — no user-controlled request host (no SSRF). Server-side `ERB` is used only on trusted config files (`authz_rules`, `amqp.yml`, seeds) and parsed with `YAML.safe_load` (no SSTI). No `system/exec/popen/Open3/shell` sinks; `instance_eval/class_eval` are class-level DSL, not request input. No `Marshal.load` and no unsafe `YAML.load`. The one reflection exception is `constantize` on params (M25, bounded).

P24 — Frontend XSS/secret hygiene is sound. User-frontend source maps are disabled in the production build (`GENERATE_SOURCEMAP=false`, `package.json`); no hardcoded secrets in frontend source; no `dangerouslySetInnerHTML` in application code (the one app-level `innerHTML` is a detached strip-tags helper, and the remaining `innerHTML` usage is inside the vendored TradingView charting bundle). Combined with the remediated prior-audit Criticals and fail-closed address validation (P8), the user frontend is in good shape. (Residual: confirm the admin dashboard build also disables source maps, and that KYC document-upload enforces a content-type/size allowlist at the uploader.)

P25 — Session and device cookies are correctly flagged. The Rails session cookie (`_barong_session`) is set `secure: Rails.env.production?, httponly: true, same_site: :strict (config/initializers/sessions_store.rb)`, and the `device_token` cookie is set `HttpOnly; SameSite=Strict; Secure` (over TLS) (`sessions.rb:386`). Both carry all three protections — not script-readable, not cross-site, TLS-only. (The separate M15 concern is the *admin frontend's* deliberately JS-readable CSRF cookie, a different mechanism.) The JWT access/refresh tokens are header-based (not cookies), stored server-side in Redis.

P26 — Frontend API call-site and sink review is clean. A static review of both React/TS apps (excluding the vendored TradingView bundle) found: API calls attach auth correctly — cookie session + `X-CSRF-Token` header + `withCredentials` (`api/requestBuilder` in both), with **no bearer token read from JS storage and no token in any URL/query string**; DOM-XSS sinks are absent (no `dangerouslySetInnerHTML/eval/new Function/document.write` in app code); every `window.location` redirect targets a static route (no param-derived open-redirect); `postMessage` is handled by the Sumsud SDK / origin-checked. The candidate sinks trace back to constants, SDK callbacks, or API data — not URL/user input. **Confirmed by automated SAST:** a Semgrep pass — 111 security rules (`p/javascript`, `p/typescript`, `p/react`, `p/secrets`) over **1,669 files** — returned **0 findings**, and a custom taint-mode ruleset (`URL/location/event.data` sources → `innerHTML/eval/location.href/window.open` sinks; sensitive web-storage; `postMessage` origin) found **no tainted source→sink flow and no token/secret in web storage**; the single flagged `message` listener validates `event.origin` against `app.shuftipro.com`. **CodeQL whole-program dataflow was also run** (CLI 2.25.6, `javascript-security-extended`, 104 queries): `exchange-castle` = 0; `frontend-ui` = 36 raw, which on triage are **34 false positives** (every high-severity `js/xss/redirect` alert dead-ends at a React-escaped `<img src={currency_icon}>` sink fed by the server-configured market-data WebSocket — CodeQL has no model for React's auto-escaping; the rest are price-display formatting and benign WS normalization), **one genuine Low** (a ReDoS validation regex, L19) and one hygiene item (a debug `console.log`, folded into L13). The deeper engine therefore **confirms no exploitable client-side XSS/redirect/injection**. *Residual: 4 files skipped/partial-parsed; the admin app opens user-content URLs via `window.open` without `noopener` (L13 tabnabbing scope; validate scheme). See `SAST-Frontend-Semgrep.md` and `SAST-Frontend-CodeQL.md`.*

Appendix: Positive Observations

P27 — Refresh-token rotation, token entropy, and JWT key-id handling are sound. Refresh is **single-use rotation**: `/refresh` calls `remove_and_generate`, which mints a new session and `flush_by_uid`'s the old refresh token (`jwt_session/session.rb:20-24`) — a stolen refresh token can't be replayed after use. All long-lived tokens use `SecureRandom` (80–128 bits: reset, device, api-key secret/kid, jti, csrf, admin whitelink) — only the short 2FA/verification codes are weak (M6). The session JWT verify path pins `algorithms: 'RS256'` as an array (`jwt.rb:32`, `Auth0 :10`) — **no RS256↔HS256 algorithm confusion**. There is **no token-header kid/jku key-selection surface**: the codec uses a single fixed keystore key, `kid` is only an API-key DB identifier (parameterized `find_by`), and the lone JWKS consumer (`Auth0`) fetches from a fixed config domain and is disabled. Credential-change revocation is partially present (password change disables API keys + service accounts — though not sessions, M28). *[Correction — this revocation matrix is **inverted**. At HEAD, password change/reset **does** revoke sessions and freeze withdrawals, but it does **not** disable API keys or service accounts: `User#disable_api_keys` (`user.rb:181`) fires only on 2FA-disable (`otp==false`) or deactivation, never on password change. **Real residual**: an attacker's API keys **survive** the victim's password-reset recovery — recovery should also inactivate the account's API keys.]*

P28 — The referral/affiliate relationship boundary is write-once and server-derived. The parent (referrer) link is set in `Member.from_payload` only when `member.ancestry.nil?` (`member.rb:501-502`) — **write-once**, so an existing referrer cannot be changed. And `referral_uid` there comes from the **verified Barong JWT payload** (the user's own signup referral, server-signed — not user-suppliable at the engine), so a user cannot set an arbitrary member as their (or anyone's) parent. Signup attribution itself is validated (`parse_refid!`: existence + uid-prefix format + a per-referrer count limit), and `referral_id` has no writer in the `resource/admin` user-update paths. Auto-earned referral rewards (`ReferralTradeReward/MiningReferralReward`) therefore derive from an immutable relationship and cannot be gamed by re-pointing; reward reads are self-scoped and redemption is balance-bounded (P22). The one residual is the share-reward recipient (L18, bounded by the caller's own earned balance — redirect/collusion, not inflation). **The no-parent edge is confirmed closed**: `referral_uid` in the bearer is the user's own (`User#as_payload`, `user.rb:321`), so a no-referrer member has `referral_uid: nil` and is never assigned a parent; `from_payload` only ever syncs the authenticating user's own member; and `member.rb:502` is the *sole* writer of `parent`. The only way to set an arbitrary member's parent is the `management/notifications` path (member-creation only), which requires a management signing JWT — gated by the same management-key trust as H4/H9, not user-reachable.

P29 — Login enforces a second factor unconditionally; a correct password alone never opens a session. `POST .../identity/sessions` (`app/api/v2/identity/sessions.rb`) requires a second factor on **both** branches after the password check: accounts with TOTP must pass `TOTPService.validate?(otp_code)` (blank/invalid rejected, `:309-311`); accounts without TOTP must supply an **expiry-bounded email/SMS one-time code** — login is refused unless `user.code_expire_at ≥ Time.now` (`:226`, `session.code_expired_at_be_expire`) and `params[:code] == user.code` (`:244`), with failed attempts incrementing `user_otp_attempts` and triggering a timed lockout (`otp_locked_out?` / `relogin_after`). **Confirmed live (2026-06-17)**: a password-correct API login was rejected with `session.code_expired_at_be_expire` — the password by itself did not create a session, demonstrating the second factor is enforced server-side, not just client-side. (Caveat: this positive concerns the *enforcement* of the second factor, which is structural and unconditional; the one-time code's entropy is tracked under M6.) **Cross-ref C5**: this positive concerns enforcement only — the very `code/verification_code` column that satisfies this second factor is disclosed for every user by C5 (the `records/:platform` leak, live-confirmed), so the secret being checked is readable by any authenticated member, materially weakening this control until C5 is fixed. *[Fixed since audit C5: the `records/:platform` endpoint was removed (319d8fe/694b567), so the 2FA `code` column is no longer disclosed and this control is no longer weakened by C5.]*

P30 — The `resource` namespace serializes users correctly — C5 is an isolated lapse, not the pattern. Auditing every endpoint in the authenticated-member `resource` namespace confirmed that `GET records/:platform` (C5) is the **only** route returning a raw, unscoped `User`. Every other User-returning endpoint is sound: `/me` and the phone flows present `current_user` through the curated `V2::Entities::UserWithFullInfo` entity, which exposes only an `otp` enabled-flag and **never** `password_digest`, `code/verification_code`, or any secret; list/collection endpoints are scoped to `current_user.*`; and `session/delete` is ownership-gated (`403 unless current_user.uid == user.uid`). So the C5 fix is a **single route**, not a pervasive rewrite. *[Fixed since audit C5: that single route was removed (319d8fe/694b567).]* (The two adjacent raw/opt-out serialization issues in the namespace are already tracked: **M8** — `/generate_code` acting on an arbitrary `user_email`; and **M10** — the api-key detail endpoint returning `secret`.)

*Caveat: positive observations describe what held at the time of review against *origin/production*; they are not guarantees.*

Appendix: Identity, 2FA & Recovery Control Surface

A coverage map of the authentication / identity control surface reviewed this engagement, with the verdict per control and the governing finding or positive. "Sound" = reviewed and no issue; "Finding" = see referenced item; "Config-dependent" = correct in code, depends on deployed configuration.

Surface	Control reviewed	Verdict	Ref
Authentication & session	Password login + session creation	Finding	C2, M18 — C2 Resolved, M18 Remediated (login response unified; 401/422 + timing oracle remain)
	Session lifetime / binding	Finding	M2 (no expiry), UA-only bind — M2 Resolved (30-min idle + 8-hr absolute; residual: UA-only bind, no IP)
	Bearer-session gate (Barong)	Finding	M1 (verify=false + cache gate)
	Gateway→engine bearer verify	Sound	P2
	JWT codec — alg / kid / jku handling	Sound	P28 (RS256 pinned; no header-key selection)
	Refresh-token rotation	Sound	P28 (single-use, old flushed)
	Token / verification-code entropy	Finding	M6 (kernel#rand 5-digit codes) — Resolved (now SecureRandom 6-digit via <code>OtpCode.generate</code>)
	Session revocation on credential change	Finding	M28 (keys/SAs revoked, sessions not) — Resolved (sessions now revoked, 319d8fe)
	CSRF enforcement	Info	I2 (gated on <code>platform=='web'</code>)
	2FA: bypass	Fixed-prod	— (OTP gen pre-fixed on prod)
2FA: recovery	TOTP validation / reuse	Finding	L11 (reuse check commented) — Resolved (reuse check re-enabled, 319d8fe)
	2FA enable/disable (user)	Finding	M19 (not password-gated) — Resolved (now password-gated, 319d8fe)
	Admin-driven 2FA disable	Finding	M24 (no admin step-up) — Resolved (acting-admin TOTP now required, 319d8fe)
	Self-service reactivation OTP	Finding	M6 (weak rand) — Resolved (now SecureRandom 6-digit)
	Enable-2FA-on-others / self-edit	Sound	P20 (blocked)
API-key scoping & IP	Capability + path scoping	Sound	P18 (default-deny, v3-scoped)
	IP-allowlist enforcement	Finding	H1 (keys off spoofable IP) — H1 Remediated (leftmost-XFF fixed), but the allowlist is still bypassable via the new <code>CF-Connecting-IP</code> spoof — see H15 (prod-confirmed live)
	Nonce replay / HMAC compare	Finding	H2, M3 — both Resolved (10s nonce + SETNX dedup; constant-time HMAC compare)
	Key-detail secret exposure	Finding	M10 (TOTP-gated, still returns secret)
KYC tier promotion	Promotion source of truth	Sound	P19 (Sumsub-authoritative)
	Webhook authenticity	Sound	P3 (HMAC)

Surface	Control reviewed	Verdict	Ref
	Applicant↔user ownership bind	Finding	M22 (incidental; control commented out) — Resolved (ownership correlation enforced, 319d8fe)
Support-driven recovery	Admin 2FA disable	Finding	M24 — Resolved
	Admin email change	Sound	P20 (superadmin-gated)
	Admin action authorization	Finding	H5 (gaps), M9; via C2 takeover — all Resolved (per-action admin_authorize!; nil-subject fixed; C2 takeover closed)

Appendix: Authorization & Object-Ownership (BOLA/IDOR) Coverage

Object-level authorization coverage for the authenticated surface, from the static IDOR/mass-assignment sweep plus targeted reads this engagement. "Self-scoped" = the handler resolves the object from `current_user` (no caller-supplied id) or filters by it; "Finding" = see ref. **Scope honesty:** this is a *representative, static* review — it has **not** been confirmed with live two-account differential testing (see the gap note), and it does not claim exhaustive per-endpoint coverage.

Endpoint group	Object access	Verdict	Ref
Profile read/update (<code>resource/profiles</code>)	<code>current_user.drafted_profile</code> , allowlisted	Self-scoped	P21
Account state (deactivate/delete/pw)	<code>current_user</code> only	Self-scoped	P21
Preference update	<code>current_user.update_columns</code> , 3 enum fields	Finding	M7 (validation-skip, bounded) — Resolved (uses <code>update</code> , validations run)
Account balances / members/me (engine)	<code>current_user</code> -scoped	Self-scoped	—
Referral reward read/redeem	<code>current_user.rewards</code> , balance-bounded	Self-scoped	P22
Referral share-reward	<code>Member.find(params[:parent_id])</code> — unvalidated recipient	Finding	L18 — Resolved (ownership assertion added)
Phone <code>generate_code</code>	acts on arbitrary <code>user_email</code>	Finding	M8 — Resolved (caller-supplied email removed)
<code>resource/users/records/:platform</code>	returns whole <code>users</code> table (incl. <code>password_digest</code> + 2FA code) — live-confirmed Total: 182 — Resolved (endpoint removed, <code>319d8fe/694b567</code>)	Finding	C5
API-key detail	returns secret (TOTP-gated)	Finding	M10
Public <code>trading_fees?uuid=</code>	unauth member fee-group by uid	Finding	L7 — Resolved (Member lookup removed)
Admin user mutate / delete	per-action authz gaps	Finding	H5, M9, M24 — all Resolved

Mass assignment

Reviewed and **largely sound**: profile updates use an explicit field allowlist (`slice(*profile_param_keys)`) and Grape `declared` returns only declared params, so caller-injected `role/level/state/uid` are dropped at every resource endpoint examined; sensitive transitions (state, level, password) are server-set on `current_user`. Residuals: the `update_columns` validation-skip (M7, bounded), and a commented-out self-promotion line (`resource/users.rb:325`) worth deleting.

Partner / affiliate boundary

Signup referral attribution (`referral_id`, Barong) accepts any existing uid as referrer, format- and existence-checked with a `referral_limit` cap — minor self-referral/farming abuse, bounded. The Peatio referral-rewards subsystem (`account/referral`, `admin/rewards|redeems|redemptions`) reads are self-scoped and redemption is balance-bounded (P22); the one boundary gap is the unvalidated share-reward recipient (L18). The admin reward/redeem-approval endpoints inherit the H5/H8 admin-authz posture and were not individually re-tested.

Appendix: Web Application Security Sink Coverage

Coverage of the classic web-application sink classes across the Rails services and the two frontends, against *origin/production*. “Clean” = swept, no instance found; “Finding” = see ref; “Verify” = not determinable from source / needs live.

Sink class	Result	Evidence / ref
XSS (frontend)	Clean	No app <code>dangerouslySetInnerHTML</code> ; one detached strip-tags <code>innerHTML</code> ; vendored chart bundle only. P24, P8 (prior crits fixed). i18n passthrough = L13
CSRF	Finding	I2 (gated on <code>platform=='web'</code>), M15 (admin token JS-readable) — I2 Resolved (platform gate removed); M15 still live (cookie→sessionStorage, still JS-readable)
SSRF	Clean	Outbound hosts fixed/config only; P23. (Minor: <code>address</code> path-interpolation into a fixed-host explorer URL)
Deserialization	Clean	No <code>Marshal/unsafe YAML.load</code> ; P23
Unsafe reflection (<code>constantize</code>)	Finding	M25 (params → <code>constantize</code> ; bounded) — Resolved (allowlist, not <code>constantize</code> on request params)
SSTI	Clean	<code>ERB</code> on trusted config only + <code>YAML.safe_load</code> ; P23
Command / eval injection	Clean	No <code>system/exec/popen/open3/shell</code> ; <code>*_eval</code> is class DSL; P23
File-upload validation	Verify	API layer caps doc count only; confirm uploader content-type/size allowlist (<code>resource/documents.rb</code>)
Open redirect	Clean	User-facing redirect built from <code>ENV['URL_HOST']</code> ; <code>callback_url</code> forwarded to provider (provider-side note); M35 signed- <code>callback_url</code> latent
Security headers	Finding	M16 (origin none, edge sets them), L15 — M16 Resolved (nginx sets full header set on both frontends); L15 Remediated (partial)
TLS posture	Finding	M21 (<code>force_ssl</code> fail-open default; HSTS observed at edge) — Resolved (<code>force_ssl</code> now defaults true)
Source maps / leaked secrets	Clean	User FE <code>GENERATE_SOURCEMAP=false</code> , no FE secrets; P24. (Verify admin FE)
Undocumented endpoints (bundle)	Verify	Source available (CRA, not Next.js); live deployed-bundle endpoint diff still pending

Net. The dangerous server-side injection classes (SSRF, SSTI, command-exec, unsafe deserialization) are **absent** — the engine inherits OpenDAX’s disciplined I/O patterns, and the custom layer did not introduce them. The one server-side sink is bounded unsafe reflection (M25). The live-pending items are the deployed-bundle endpoint diff and the KYC-upload allowlist confirmation; the standing CSRF/header/TLS findings (I2, M15, M16, M21) are unchanged by this sweep.

Appendix: Authorization / RBAC Model

Extracted from `config/seeds.yml` (the seeded `Permission` rows) and the resolution logic in `lib/barong/authorize.rb:162-167`. Authorization is gateway-enforced per request: `permissions.select!{ role match && verb match && @path.starts_with?(rule.path) }`, then deny if **any** matched rule is `DROP` or if **no** matched rule is `ACCEPT`.

Resolution is fail-safe (positive): a `DROP` overrides any `ACCEPT`, and the absence of an explicit `ACCEPT` denies (default-deny). So the broad grants below are correctly clipped by their `DROP` exceptions.

Role	Scope (ACCEPT)	Notable restrictions / grants
superadmin	<code>all api/v2, all admin</code> — full	unrestricted (incl. roles & permissions mgmt)
admin	<code>all api/v2, all admin</code>	DROP: <code>admin/permissions</code> , <code>admin/users/role</code> , <code>admin/activities/admin</code> — cannot manage roles/permissions (good separation from superadmin)
compliance	<code>all admin/users</code> + financial/KYC read	full user/KYC management; read-only on financials
accountant	financial read + <code>adjustments/new</code>	can post ledger adjustments; can manage <code>admin/abilities</code> ; DROP: <code>role/permissions</code>
technical	infra: <code>all admin/{wallets,markets,blockchains,currencies,trading_fees,withdraw_limits,restrictions}</code> , order cancel	broad infra write; can manage <code>admin/abilities</code>
support	admin read + <code>post admin/users/update</code>	holds the user-update grant that powers M24/M28 (disable 2FA, change state/email); can manage <code>admin/abilities</code>
broker / maker / sa_maker / trader	trading / market-making scopes	FIX/finex market + order surfaces (trading roles)
member	<code>api/v2/peatio/account, market, barong/resource</code>	default authenticated user

Observations

- Denylist-on-broad-allow (refines I1).** admin/superadmin are granted `all api/v2 accept` — *everything* under the prefix — and security relies on the **completeness of the DROP list** for sensitive sub-paths. Because matching is `starts_with?`, a newly-added sensitive endpoint with **no corresponding DROP** is silently in-scope for admin. Audit the DROP set whenever admin endpoints are added; prefer explicit per-resource ACCEPTs over a blanket `api/v2` grant.
- Two enforcement layers (both server-side).** Beyond this gateway path-RBAC there is a second, in-handler *action*-based layer: `admin_authorize!(*) → AdminAbility.new(current_user).authorize!(*)` (`api/v2/utils.rb:54`), with per-role CanCanCan rules in `app/models/abilities/{role}.rb` (e.g. `authorize! :create, Wallet`). The `admin/abilities` endpoint is **read-only** (no write route) and the role→ability map is defined in code, so the `all admin/abilities accept` grant is harmless (read) and **cannot be used to self-escalate** — a positive.
- Ledger adjustments are correctly maker-checker (positive).** `accountant` holds `:create, Adjustment` (creates a *pending* adjustment) but **only superadmin holds `:manage`** (the `:write/accept` ability); `admin` is read-only. So the creator cannot self-approve a ledger credit — a genuine two-person control, with the approver recorded as `validator`.
- Excessive operational-role privilege over fund config (M29).** *[Partly fixed since audit: operational roles were downgraded (blanket `:manage/:destroy` and `DefiWallet` removed), but `technical` still holds `:write` on `RewardSetting/WithdrawLimit/Wallet` — the residual money-lever remains (ties M34).]* The action-layer grants `technical:manage, Wallet` (address / `secret` / Fireblocks vault id) and `:manage, WithdrawLimit` — fund redirection from a non-superadmin role; and `accountant :create, Deposits::Fiat` + `:manage, ColdWithdraw` — unilateral

money surfaces lacking the adjustment maker-checker. [Correction — these are the **pre-remediation** grants; at HEAD they are narrower: *accountant* = *:read, ColdWithdraw* + *:read, WithdrawLimit* (**cannot dispatch cold-wallet withdrawals**), retaining only *:create, Deposits::Fiat*; and *technical* is *:write* (not *:manage*) on *Wallet/WithdrawLimit*. The genuine residual is *technical :write* on *RewardSetting/Wallet* (ties M34), per the M29 update above.] Segregate fund-custody config to superadmin / multi-party approval.

Verify against the **deployed** *Permission* table — seeds can be edited post-deploy; the matrix above is the seeded baseline.

Appendix: Deposit / Withdraw State-Machine Model

The money-safety core. Extracted from the AASM definitions in `app/models/withdraw.rb:84+` and `deposit.rb:51+`, annotating each transition with its **fund side-effect** and the **lock held**. The point of modeling it: a fund-moving transition that isn't serialized on the row can be re-entered (double-credit / double-debit).

Withdraw (`aasm whiny_transitions: false`)

Event	From → To	Fund effect
<code>accept</code>	prepared → accepted	<code>lock_funds</code> (balance→locked)
<code>done</code>	deposited/accepted → succeed	<code>unlock_and_sub_funds</code> (DEBIT)
<code>success</code>	confirming/deposited/errored/under_review → succeed	<code>unlock_and_sub_funds</code> (DEBIT)
<code>cancel</code>	prepared/accepted/skipped → canceled	<code>unlock_funds</code> (refund, unless from prepared)
<code>reject</code>	to_reject/accepted/confirming/under_review → rejected	<code>unlock_funds</code> (refund)
<code>fail</code>	processing/confirming/skipped/errored/under_review → failed	<code>unlock_funds</code> (refund)
<code>err</code>	processing → errored	none — funds stay locked (manual-retry state)
process/load/review/dispatch/deposited/skip/locked	(routing)	none

Deposit

Event	From → To	Fund effect
<code>accept</code>	submitted/release/locked → accepted	<code>account.plus_funds</code> (CREDIT)
reject / cancel	submitted → rejected/canceled	none
release / locked / skip / process / collect...	(routing / sweep)	none (sweep moves to hot/cold, not user balance)

Observations

- **Fund arithmetic is row-locked (positive).** `lock/unlock/plus/sub_funds` all run inside `account.with_lock { ... }` (`account.rb:72-124`) — the balance columns can't be corrupted by concurrent arithmetic.
- **Refund-on-failure is complete (positive).** Every normal failure transition refunds the locked amount — `cancel`, `reject`, and `fail` all call `unlock_funds`. No stuck/lost funds on the standard failure paths. (Residual: `errored` leaves funds *locked* pending manual retry — not lost, but a withdraw stuck in `errored` ties up the balance indefinitely.)
- **Transitions are not row-serialized.** The `with_lock` protects the `account` arithmetic, but the deposit/withdraw **row is not lock!'d for the duration of the transition**, and the guards are evaluated before the side-effect. Two concrete consequences: **(deposit)** concurrent/duplicate `accept!` both credit (H6; reachable via the C1 webhook's retries); **(withdraw)** the terminal `succeed` state has **two debit events** (`done` and `success`) reachable from overlapping source states (`deposited/accepted`) — if a worker-driven `done` and `success` race on the same withdraw, both pass their guard and both call `unlock_and_sub_funds` → **double-debit**. **Fix:** `row.lock!` at the top of every fund-moving transition and re-check the state under the lock; collapse the duplicate `done/success` debit paths. (M12, sharpened; H6.)

Appendix: Vault Secret Channel & ShuftiPro KYC Callback

A. Vault secret channel (Barong ↔ HashiCorp Vault)

Vault is the trust anchor for the two most sensitive secret classes in the auth service, accessed through a **custom fork** of `vault-rails` (`plugins-vault-rails/`):

Secret	Vault engine / path	Code
API-key signing secret	transit (<code>vault_attribute</code>)	<code>app/models/api_key.rb:16</code>
TOTP 2FA seed + code validation	TOTP engine	<code>app/services/totp_service.rb:25,44</code>
Generic KV secrets	logical KV	<code>app/services/secret_storage.rb:41-53</code>

What holds (positive): TOTP seeds live *inside* Vault’s TOTP engine and never touch the application DB; the forked `EncryptedModel`/transit logic is otherwise faithful to upstream; `config.enabled` is correctly true in production; a token-renewal thread keeps the lease fresh.

What does not (findings):

- **M31** — `enabled = Rails.env.production?` (`vault.rb:6`) + in-memory key `MD5(path+key+context)`, no IV (`rails.rb:203-209,180-183`): in any non-prod env the API-key secret is at rest under a publicly-recomputable key.
- **L22** — `puts request_opts` in `transform_decode` (`rails.rb:167`) prints FPE plaintext to logs (ties M26). **L23** — `require "pry"` at the top of the production crypto lib (`encrypted_model.rb:1`).

Live-verify: the production `VAULT_ADDR` scheme (`http` vs `https`) and whether the Vault service is network-segmented from application pods — these decide whether H14 is active-MITM or passive-sniff.

B. ShuftiPro KYC callback (vs the Sumsb control)

Endpoint `POST /api/v2/barong/public/shuftipro/kyc` (`app/api/v2/public/general.rb:106`). **It is not a blind-trust forgery.** Unlike the legacy Jumio callback, the status is *not* trusted from the POST body — the inline update is commented out (`shuftipro_service.rb:29-31`) and the worker re-fetches the authoritative status server-side: `check_status` → `POST https://api.shuftipro.com/status` with Basic auth over TLS (`lib/shufti_pro/merchant.rb:27, client.rb:24`). That re-fetch is the control that de-risks it from Critical.

Residual gaps (defense-in-depth, Low→Medium — documented here, not yet a numbered finding):

Control	Sumsb <code>/sumsub/kyc</code>	ShuftiPro <code>/shuftipro/kyc</code>
Webhook signature verified	Yes — HMAC <code>X-Payload-Digest</code> (<code>general.rb:78-86</code>)	No — no check; no signature helper in <code>lib/shufti_pro</code>
Caller-to-user binding	applicant id	<code>params['reference'] = applicant PK = "#{uid}_#{ms}"</code> (enumerable , <code>shuftipro_service.rb:58</code>)
Status trusted from body	partially	No (re-fetched) — the mitigating control
Replay / idempotency / rate-limit	—	None

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer's risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.