# **Security Remediation Plan & Work Orders**

## **Delta review — new code push (Sep 18, 2025)**

**Scope of check:** Compared this document’s acceptance criteria to the latest repo (liftedvfo-main). Searched for auth storage, HTML rendering, uploads, CORS, admin endpoints, and token-at-rest patterns.

**Summary:** Most Priority 0 items remain **unfixed**; CORS became **wider**; and an additional unsanitized HTML render point exists.

### **Status by blocker**

* **0.1 Secure cookie sessions (browser tokens):** *No change.* Still using localStorage for access\_token and Google tokens (frontend/src/apiClient.ts, frontend/src/pages/Login.tsx, frontend/src/services/googleCalendarService.ts).
* **0.2 Encrypt provider refresh tokens at rest:** *No change.* Plaintext columns google\_access\_token, google\_refresh\_token exist in backend/app/models/user.py; migration backend/app/db/migrations/add\_google\_auth\_fields.py persists these.
* **0.3 Remove or sanitize raw HTML:** *Worse surface than before.* In addition to EditableSection.tsx, frontend/src/pages/PublicBlog.tsx renders content with dangerouslySetInnerHTML without sanitization.
* **0.4 Upload limits & streaming:** *No change.* /api/chat/upload-and-index and legal document upload accept files without MIME/type allowlist or size caps (backend/app/api/chat.py, backend/app/api/legal.py).
* **0.5 CORS & SECRET\_KEY:** *Regressed on CORS.* backend/app/main.py now allows **more** origins (added https://liftedvfo-frontend.onrender.com, https://app.liftedvfo.io) with allow\_credentials=True and allow\_methods=["\*"]. Default SECRET\_KEY still present in backend/app/core/config.py.
* **0.6 Admin header token:** *No change.* backend/app/api/admin\_tasks.py still requires X-Admin-Token shared secret for migrations.

### **Additional findings (new since prior write‑up)**

* **Hardcoded Google OAuth client ID fallback:** frontend/src/providers/GoogleAuthProvider.tsx includes a literal fallback client ID when VITE\_GOOGLE\_CLIENT\_ID is absent. *Required change:* remove fallback; require env var and fail fast.
* **Configuration probe endpoint present:** backend/app/api/test\_config.py is still enabled and exposed under /api/test/config. *Required change:* remove in prod or restrict to SuperAdmin + internal network.

### **Suggested doc updates (non‑destructive)**

To avoid changing your existing sections, add these notes when the vendor picks up the work:

* For **0.3**, include frontend/src/pages/PublicBlog.tsx in the **Files to update** list.
* For **0.5**, call out that widening the origin list increased risk; prod must allow only the official UI origin.
* Add a **Priority 1** item: *Remove hardcoded Google Client ID fallback* with acceptance criteria: “VITE\_GOOGLE\_CLIENT\_ID is required; build fails if missing; no literal client IDs in repo history.”

## **Priority 0 blockers (must be fixed before any prod traffic)**

### **0.1 Replace browser tokens with secure cookie sessions**

**Required change**

* Complete OAuth on the backend; never expose provider refresh tokens to the browser.
* Issue HTTP-only, Secure, SameSite session cookies; remove any localStorage token usage and configure Axios with withCredentials: true.
* Implement server-side logout to clear the cookie.

**Why this matters**

* Tokens in localStorage are readable by any injected script. A single XSS bug equals full account takeover.
* HTTP-only cookies are not accessible to JavaScript and combined with SameSite reduce CSRF and token exfiltration risk.
* Keeping OAuth on the backend shrinks the blast radius and simplifies revocation and auditing.

**Acceptance criteria (bulleted for readability)**

* No references to localStorage.getItem('token') or similar in the repo.
* Login sets an HTTP-only cookie. DevTools shows HttpOnly=true, Secure=true, SameSite=Strict or Lax.
* API rejects requests without the cookie. Frontend sends withCredentials on calls.
* Logout clears the cookie server side.

**Files to update (from audit)**

* Frontend: frontend/src/apiClient.ts, frontend/src/pages/Login.tsx, frontend/src/services/googleCalendarService.ts, frontend/src/providers/GoogleAuthProvider.tsx
* Backend: backend/app/api/api.py, backend/app/api/google\_auth.py

**Backend sketch (FastAPI)**

****# set cookie after successful login

from fastapi import Response

resp = Response(status\_code=204)

resp.set\_cookie(

key="session",

value=create\_signed\_session(user\_id),

httponly=True,

secure=True,

samesite="lax",

max\_age=60\*60\*8,

)

return resp

**Frontend sketch (Axios)**

****// apiClient.ts

import axios from 'axios'

export const api = axios.create({ baseURL: import.meta.env.VITE\_API\_URL, withCredentials: true })



### **0.2 Encrypt provider refresh tokens at rest**

**Required change**

* Encrypt provider tokens using application-layer envelope encryption (DEK wrapped by KMS); store only ciphertext and IV.
* Add a data migration to backfill encrypted values and drop plaintext columns after verification.
* Define and test a key rotation procedure.

**Why this matters**

* A database leak of plaintext refresh tokens lets attackers mint new access tokens indefinitely.
* Application-layer encryption with KMS ensures stolen rows are useless without keys, supporting compliance and breach notification minimization.
* Clear operational rotation procedures reduce long-term key exposure.

**Acceptance criteria (bulleted for readability)**

* Columns that store provider tokens are encrypted at the application layer before insert.
* Key rotation documented. Rotation runbook provided.
* Raw tokens never appear in logs, traces, or debug output.

**Migration approach**

* Add new columns google\_refresh\_token\_enc, google\_refresh\_token\_iv.
* Backfill: read plaintext, encrypt, write encrypted, then null the plaintext column.
* Drop plaintext columns after verification.

**Pseudocode**

****ciphertext, iv = kms\_encrypt(dek, raw\_refresh\_token)

model.google\_refresh\_token\_enc = ciphertext

model.google\_refresh\_token\_iv = iv



### **0.3 Remove raw HTML rendering or sanitize it**

**Required change**

* Remove direct uses of dangerouslySetInnerHTML for user content; render via a sanitizer (e.g., DOMPurify) or switch to Markdown rendering.
* Add unit tests that assert script and inline-handler payloads are stripped.

**Why this matters**

* Rendering unsanitized HTML enables cross-site scripting, which can steal sessions, change wire instructions, or inject fake financial data.
* Sanitization or Markdown eliminates script execution while preserving formatting.

**Acceptance criteria (bulleted for readability)**

* No direct uses of dangerouslySetInnerHTML remain for user content.
* Unit tests prove that <script> and onerror payloads are stripped.

**Files to update**

* Frontend: frontend/src/components/modules/EditableSection.tsx and any similar component.

**Sketch**

****import DOMPurify from 'dompurify'

export function SafeHtml({ html }: { html: string }) {

return <div dangerouslySetInnerHTML={{ \_\_html: DOMPurify.sanitize(html, { USE\_PROFILES: { html: true } }) }} />

}



### **0.4 Lock down uploads by size and type, stream safely**

**Required change**

* Enforce a MIME allowlist and a 20 MB size cap; stream uploads to disk.
* Reject archives/multipart bombs; return HTTP 413/415 for over-size or disallowed types.
* Do not read entire files into memory for analysis.

**Why this matters**

* Unbounded uploads enable resource exhaustion, unexpected cloud costs, and parser exploits (e.g., zip bombs, malformed PDFs).
* Type allowlists and size caps prevent common abuse paths and protect the embedding/RAG pipeline.

**Acceptance criteria (bulleted for readability)**

* Requests over 20 MB return HTTP 413.
* Non-allowlisted MIME types return HTTP 415.
* Endpoint never reads the entire file into memory for large files.

**Backend sketch**

****ALLOWED\_TYPES = {"application/pdf", "application/vnd.openxmlformats-officedocument.wordprocessingml.document"}

MAX\_BYTES = 20 \* 1024 \* 1024



### **0.5 CORS and secret key hardening**

**Required change**

* Restrict CORS in production to the single, official frontend origin.
* Replace allow\_methods=["\*"] with only required methods; set allow\_credentials only if needed.
* Require a strong, non-default SECRET\_KEY from a secrets manager; fail fast on missing secrets in prod.

**Why this matters**

* Wide or incorrect CORS lets rogue sites call your API as the user and read responses.
* Weak or default secrets enable JWT forgery and session impersonation.

**Acceptance criteria (bulleted for readability)**

* Process exits on prod if SECRET\_KEY or DATABASE\_URL is missing.
* allow\_origins contains only the prod frontend domain in prod.

**Files**

* backend/app/core/config.py, backend/app/main.py

### **0.6 Replace any admin header token with real RBAC**

**Required change**

* Remove the shared X-Admin-Token header mechanism.
* Require authenticated sessions with an admin role; add IP allowlist or VPN for admin routes.
* Add rate limiting and audit logging for all admin actions.

**Why this matters**

* Shared header tokens leak via logs or screenshots and provide full admin access with no audit trail.
* Proper RBAC enforces least privilege and enables accountability.

**Acceptance criteria (bulleted for readability)**

* Admin endpoints return 401 without a valid session and admin role.
* No code references to ADMIN\_TASKS\_TOKEN remain.

**Files**

* backend/app/api/admin\_tasks.py

## **Priority 1 (close behind)**

### **1.1 Brute force protection and admin 2FA**

**Required change**

* Add per-username and per-IP rate limiting to /token and any password login endpoint.
* Enable 2FA for admin/superadmin (WebAuthn preferred; TOTP acceptable).

**Why this matters**

* Credential stuffing and password reuse are common. Rate limits blunt automated attacks.
* 2FA dramatically reduces successful compromises of privileged accounts.

**Acceptance criteria (bulleted for readability)**

* 10 wrong attempts per 15 minutes blocks the username and IP.
* Admin accounts can enroll a WebAuthn key or TOTP.

### **1.2 RAG and AI safety guardrails (if AI features are present)**

**Required change**

* Pin a system prompt; never treat retrieved content as instructions.
* Restrict tool access behind an allowlist; return source-cited answers.
* Add tests that attempt prompt-injection/exfiltration and verify containment.

**Why this matters**

* Prompt injection can coerce AI features to exfiltrate secrets or perform unsafe actions.
* Source-grounded responses with strict tool allowlists prevent data leakage and bad financial guidance.

**Acceptance criteria (bulleted for readability)**

* Test document that tries to exfiltrate secrets fails to do so.
* Responses include citations when they rely on retrieved content.

### **1.3 Schema migrations via Alembic**

**Required change**

* Introduce Alembic migrations; remove all schema DDL from application startup.
* Provide versioned upgrade/downgrade scripts and document rollback.

**Why this matters**

* Startup DDL causes race conditions, environment drift, and risky rollbacks.
* Versioned migrations give repeatability, audit trails, and safe roll-forward/rollback.

**Acceptance criteria (bulleted for readability)**

* alembic/versions/ contains migration files for all changes.
* App starts without running DDL.

### **1.4 Security headers and CSP at the edge**

**Required change**

* Serve the frontend via a reverse proxy that sets CSP, HSTS, X-Content-Type-Options, and Referrer-Policy.
* Start CSP in report-only, fix violations, then enforce.

**Why this matters**

* CSP blocks inline script execution and many classes of XSS.
* HSTS enforces HTTPS and prevents downgrade attacks. Other headers reduce content sniffing and referrer leaks.

**Acceptance criteria (bulleted for readability)**

* Report-only CSP shows zero violations under normal use.
* Strict CSP enabled after a short bake-in.

**NGINX example**

****add\_header Content-Security-Policy "default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:; connect-src 'self' https://api.example.com; frame-ancestors 'none'" always;

add\_header Strict-Transport-Security "max-age=15552000; includeSubDomains" always;

add\_header X-Content-Type-Options nosniff always;

add\_header Referrer-Policy strict-origin-when-cross-origin always;



### **1.5 Observability, health, and error reporting**

**Required change**

* Implement /healthz and /readyz endpoints (no secrets in output).
* Switch to structured JSON logs with request IDs; wire error reporting (e.g., Sentry).

**Why this matters**

* Without health checks and telemetry, outages and partial failures go undetected.
* Structured, redacted logging speeds incident response without leaking sensitive data.

**Acceptance criteria (bulleted for readability)**

* Liveness and readiness probes respond in under 50 ms and do not leak secrets.
* Errors are captured with request IDs and redacted context.

## **Priority 2 (round out the posture)**

### **2.1 PII masking and selective encryption**

**Required change**

* Identify sensitive columns; mask in logs and admin UIs; encrypt where justified by risk.
* Prohibit logging of raw PII by policy and lint checks.

**Why this matters**

* PII minimization and masking reduce breach impact and help meet privacy obligations.
* Targeted column encryption limits insider and lateral movement risk.

**Acceptance criteria (bulleted for readability)**

* Configured list of sensitive fields is masked in logs and admin UIs.

### **2.2 Dependency and image scanning in CI**

**Required change**

* Add SAST, dependency audits, and container scans to CI; publish SBOM artifacts per release.
* Fail the pipeline on high/critical CVEs.

**Why this matters**

* Most compromises enter through third-party code or outdated images.
* Automated scans and SBOMs make exposure visible and auditable before release.

**Acceptance criteria (bulleted for readability)**

* CI fails on high CVEs. SBOM is attached to release artifacts.

**GitHub Actions sketch**

****name: security

on: [push, pull\_request]

jobs:

py-audit:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-python@v5

with: { python-version: '3.11' }

- run: pip install -r backend/requirements.txt pip-audit bandit

- run: bandit -r backend/app

- run: pip-audit -r backend/requirements.txt

node-audit:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-node@v4

with: { node-version: '20' }

- run: cd frontend && npm ci && npm audit --omit=dev



### **2.3 Docker hardening**

**Required change**

* Convert Dockerfiles to multi-stage builds; run as a non-root user.
* Minimize the base image and strip build tools from the final image.

**Why this matters**

* Root containers and large images expand the attack surface and ease privilege escalation.
* Minimal, non-root images reduce vulnerabilities and improve supply chain hygiene.

**Acceptance criteria (bulleted for readability)**

* USER app or similar is present. Container runs without root.
* Final image size is reduced and passes Trivy with no high CVEs.

## **PR checklist (attach to each remediation PR)**

* Includes unit tests for the fix
* Includes docs or code comments for operators
* No secrets or tokens in code or logs
* Threat model notes updated if behavior changes
* Rollback plan included in the PR description

## **Test plan for acceptance**

* Manual tests for login, logout, cookie flags, and blocked localStorage access
* Upload of allowed and disallowed files and sizes
* Script and event handler XSS payloads are sanitized
* Rate limit behavior on auth routes
* CSP report-only review and final enablement
* DB inspection shows encrypted values, not plaintext, for provider tokens

## **Stop-ship rules**

* Tokens in browser storage
* Unencrypted provider tokens in the database
* Raw HTML rendering of user content without a sanitizer
* Upload endpoints without size and type limits
* Admin access via a shared header token

## **Exhibit A: Evidence pack (paths & excerpts)**

### **A1. Browser-stored tokens (finding 0.1)**

**Paths:**

* frontend/src/apiClient.ts
* frontend/src/pages/Login.tsx
* frontend/src/services/googleCalendarService.ts

**Excerpts:**

**** 8:

9: apiClient.interceptors.request.use(

10: (config) => {

11: const token = localStorage.getItem('access\_token');

12: if (token) {

13: config.headers['Authorization'] = `Bearer ${token}`;

14: }



 22: 'Content-Type': 'application/x-www-form-urlencoded',

23: }

24: });

25: localStorage.setItem('access\_token', response.data.access\_token);

26: // Store user name from response, fallback to email prefix

27: const userName = response.data.user\_name || email.split('@')[0];

28: localStorage.setItem('user\_name', userName);

---

25: localStorage.setItem('access\_token', response.data.access\_token);

26: // Store user name from response, fallback to email prefix

27: const userName = response.data.user\_name || email.split('@')[0];

28: localStorage.setItem('user\_name', userName);

29: // Store role from response

30: const role = response.data.role || 'Client';

31: localStorage.setItem('role', role);



 35: private refreshAccessToken() {

36: const token = localStorage.getItem('google\_access\_token');

37: if (token && token !== this.accessToken) {

38: this.accessToken = token;

39: }

40: }

...

134: Authorization: `Bearer ${this.accessToken}`,



### **A2. Plaintext provider tokens at rest (finding 0.2)**

**Path:** backend/app/models/user.py

**Excerpt:**

**** 21: picture\_url = Column(String, nullable=True)

22: is\_active = Column(Boolean, default=True)

23: created\_at = Column(DateTime, default=datetime.utcnow)

24: google\_access\_token = Column(String, nullable=True)

25: google\_refresh\_token = Column(String, nullable=True)



### **A3. Unsanitized HTML rendering (finding 0.3)**

**Paths:**

* frontend/src/components/modules/EditableSection.tsx
* frontend/src/pages/PublicBlog.tsx

**Excerpts:**

**** 96: {isHtml && typeof content === 'string' ? (

97: <div dangerouslySetInnerHTML={{ \_\_html: content }} />

98: ) : (



 56: <div

57: style={{ lineHeight: '1.8', fontSize: '18px' }}

58: dangerouslySetInnerHTML={{ \_\_html: post.content.replace(/\n/g, '<br/>') }}

59: />



### **A4. Upload endpoints without size/type guardrails (finding 0.4)**

**Paths:**

* backend/app/api/chat.py (/api/chat/upload-and-index)
* backend/app/api/legal.py (entity document uploads)

**Excerpts:**

**** 76: @router.post("/upload-and-index")

77: async def upload\_and\_index\_document(

78: file: UploadFile = File(...),



 12: @router.post("/entities/{entity\_id}/documents/", response\_model=Document)

13: def upload\_document\_for\_entity(

14: entity\_id: int,

15: file: UploadFile = File(...),



### **A5. CORS posture and default secret key (finding 0.5)**

**Paths:**

* backend/app/main.py (CORS)
* backend/app/core/config.py (SECRET\_KEY default)

**Excerpts:**

****allow\_origins=[

"http://localhost:3000",

"http://localhost:5173",

"http://localhost:5174",

"https://agentiq-vfo-frontend.onrender.com",

"https://liftedvfo-frontend.onrender.com",

"https://app.liftedvfo.io",

],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)



 SECRET\_KEY: str = "your-secret-key-here"



### **A6. Admin endpoints protected by shared header token (finding 0.6)**

**Path:** backend/app/api/admin\_tasks.py

**Excerpt:**

****@router.post("/migrations/google-auth")

def run\_google\_auth\_migration(x\_admin\_token: str | None = Header(default=None)):

expected = os.getenv("ADMIN\_TASKS\_TOKEN", "")



### **A7. Configuration probe endpoint exposed**

**Path:** backend/app/api/test\_config.py

**Excerpt:**

****@router.get("/config")

async def test\_config():

return {

"google\_client\_id\_configured": bool(os.getenv("GOOGLE\_CLIENT\_ID")),

...

}



### **A8. Hardcoded Google OAuth client ID fallback**

**Path:** frontend/src/providers/GoogleAuthProvider.tsx

**Excerpt:**

****const clientId = import.meta.env.VITE\_GOOGLE\_CLIENT\_ID || '3...apps.googleusercontent.com';

## **Delivery and sign-off**

* Deliver fixes as small PRs grouped by priority item
* Include migration scripts and rollback steps where relevant
* Provide a one-page summary of changes and how to validate them
* We will test in our staging environment and sign off in writing when all acceptance criteria are met