Informe técnico (auditoría + remediación) – NOVACORE Core Bancario (AWS EC2)

Este documento explica **cómo** se materializa cada hallazgo, muestra **fragmentos de código** (archivo y líneas) y propone **instrucciones de corrección** orientadas a un entorno AWS EC2 detrás de ALB/WAF. Las instrucciones son defensivas y no incluyen procedimientos de explotación.

1) Secreto/API key hardcodeado en script [CRÍTICO]

Ubicación: scripts/test-opm-curl.sh (línea 8)

Cómo ocurre / por qué existe: El script incluye un valor literal de API_KEY dentro del repositorio. Cualquier copia del repo conserva ese secreto.

Evidencia (extracto):

```
0008: API_KEY="cba28c3936558c5bf851d5d67d9d36a1fb69b27a717d6fe4ecd759215e7ef632"
```

Impacto bancario: Compromiso de credenciales de proveedor; acceso no autorizado a APIs de pagos/operación según permisos del key.

Instrucciones de corrección (paso a paso):

• Rotar/revocar inmediatamente la key en el panel del proveedor (OPM) y registrar el incidente.

• Eliminar el valor del repo: cambiar el script para leer desde env var (ej. OPM_API_KEY) y abortar si falta.

• Eliminar el secreto del historial Git si el repo se compartió (ej. filter-repo/BFG) y forzar rotación.

• Agregar escaneo de secretos en CI (gitleaks/trufflehog) y bloquear merges si detecta patrones.

2) Token de sesión persistido en localStorage (riesgo XSS) [CRÍTICO]

Ubicación: src/context/AuthContext.tsx (líneas 56, 136, 166, 199, etc.)

Cómo ocurre / por qué existe: El frontend guarda la sesión (incluyendo token) en localStorage y lo reutiliza para construir Authorization Bearer. Cualquier ejecución de JS en el navegador (XSS/3rd-party) puede leer ese valor.

Evidencia (extracto):

```
0056: const sessionStr = localStorage.getItem('novacorp_session');
0136: localStorage.setItem('novacorp_session', JSON.stringify(session));
0166: 'Authorization': `Bearer ${session.token}`,
0199: localStorage.setItem('novacorp_session', JSON.stringify(session));
```

Impacto bancario: Toma de cuenta (account takeover) si ocurre XSS o dependencia vulnerable; acceso a balances y operaciones con sesión robada.

Instrucciones de corrección (paso a paso):

• Migrar a sesión cookie-only: eliminar guardado/lectura de token en localStorage (mantener solo datos no sensibles en memoria).

• Eliminar el uso de Authorization Bearer desde el navegador: depender de cookie httpOnly enviada automáticamente.

• Ajustar el cliente fetch/axios para NO adjuntar tokens manualmente; usar credenciales por cookie (same-site).

• Endurecer CSP (con nonces) y headers para contener XSS; revisar dependencias.

3) Login devuelve token en JSON pese a usar cookie httpOnly [CRÍTICO]

Ubicación: src/app/api/auth/login/route.ts (líneas 301–370 aprox.; token incluido en response en línea 359)

Cómo ocurre / por qué existe: El endpoint crea un token de sesión y lo setea como cookie httpOnly, pero también lo incluye en el cuerpo JSON ("backward compatibility"). Eso expone el token al JS del navegador.

Evidencia (extracto):

```
0301: // Create session token
```

```
0302: const token = crypto.randomUUID();
0356: // SECURITY FIX: Create response with httpOnly secure cookie
0359: token, // Still include token in response for backward compatibility
0366: name: 'novacorp_token',
0368: httpOnly: true,
0370: sameSite: 'strict',
```

Impacto bancario: Eleva el impacto de cualquier XSS: el atacante no necesita robar cookies httpOnly, basta leer el JSON de login o el storage.

Instrucciones de corrección (paso a paso):

• Eliminar el token del JSON de respuesta en producción (y preferentemente en todos los entornos).

• Si hay clientes legacy: crear un modo legacy separado (otro endpoint o header) y deshabilitarlo en prod por default.

• Añadir tests que fallen si el response incluye 'token' para login en ENV=production.

4) Webhook secret mismatch no bloquea procesamiento [CRÍTICO]

Ubicación: src/app/api/webhooks/deposit/route.ts (líneas 72–88)

Cómo ocurre / por qué existe: El handler compara el secreto (timingSafeEqual) pero en caso de mismatch solo hace console.warn y continúa el flujo.

Evidencia (extracto):

```
0072: // Optional: Validate webhook secret if configured
0077: if (webhookSecret && receivedSecret) {
0081: if (secretBuffer.length !== receivedBuffer.length ||
0082:     !crypto.timingSafeEqual(secretBuffer, receivedBuffer)) {
0083:   console.warn('Webhook secret mismatch');
0088: }
```

Impacto bancario: Permite que eventos no autenticados sigan consumiendo lógica, recursos y caminos de negocio; riesgo de falsos abonos si otras validaciones fallan o se degradan.

Instrucciones de corrección (paso a paso):

• Convertir mismatch en condición de rechazo: retornar 401/403 y NO continuar procesamiento.

• Registrar auditoría del intento (sin PII) y métricas/alertas por tasa de mismatch.

• Asegurar que WAF/ALB restrinjan el origen (IPSet) para reducir superficie.

5) Validación de firma RSA del webhook puede omitirse por variable de entorno [CRÍTICO]

Ubicación: src/app/api/webhooks/deposit/route.ts (líneas 94–121)

Cómo ocurre / por qué existe: La verificación criptográfica se desactiva si SKIP_WEBHOOK_SIGNATURE_VALIDATION='true'. En producción, un error de configuración o un cambio de env podría desactivar la validación.

Evidencia (extracto):

```
0094: const skipSignatureValidation = process.env.SKIP_WEBHOOK_SIGNATURE_VALIDATION === 'true';
0096: if (!skipSignatureValidation) {
0097:   const signatureValid = await validateOpmSignature(body);
0116: } else {
0117:   console.warn('=== WEBHOOK SIGNATURE VALIDATION SKIPPED ===');
```

Impacto bancario: Si se desactiva la firma, el sistema confía en controles no criptográficos (IP/headers) y aumenta riesgo de eventos falsificados.

Instrucciones de corrección (paso a paso):

• Eliminar el bypass en builds de producción: fallar el arranque si ENV=production y esa variable está activa.

• Separar lógica de staging vs prod: el código de bypass no debe compilarse en prod (feature flag a nivel build).

• Asegurar obtención y rotación de la public key del proveedor; almacenar en Secrets Manager.

6) Conexión a DB con TLS sin verificación de certificado [CRÍTICO]

Ubicación: src/lib/db.ts (líneas 10–16)

Cómo ocurre / por qué existe: La configuración ssl define rejectUnauthorized:false, aceptando certificados no confiables.

Evidencia (extracto):

```
0010: ssl: {
0012: // rejectUnauthorized: false is acceptable here because:
0016: rejectUnauthorized: false,
```

Impacto bancario: Riesgo de MITM o interceptación si la topología/red cambia o hay rutas no estrictamente privadas; incumplimiento de prácticas de seguridad bancaria.

Instrucciones de corrección (paso a paso):

• Configurar CA oficial del motor (RDS/Aurora) y activar rejectUnauthorized:true.

• Validar que DATABASE_URL use sslmode=verify-full (cuando aplique) y hostname correcto.

• Agregar verificación en CI: bloquear merges si aparece rejectUnauthorized:false en prod config.

7) Listado de transacciones sin scope obligatorio de tenant/empresa [CRÍTICO]

Ubicación: src/lib/db.ts (líneas 1310–1349)

Cómo ocurre / por qué existe: listTransactions permite clabeAccountId opcional; si se llama sin ese filtro, el WHERE no fuerza tenant/company y podría devolver transacciones globales.

Evidencia (extracto):

```
1310: export async function listTransactions(params: { ... clabeAccountId?: string; ... })
1344: (${type}::text IS NULL OR type = ${type})
1346: AND (${clabeAccountId}::text IS NULL OR clabe_account_id = ${clabeAccountId})
1347: ORDER BY created_at DESC
```

Impacto bancario: Riesgo de fuga masiva de información financiera si un endpoint omite pasar clabeAccountId/companyId.

Instrucciones de corrección (paso a paso):

• Cambiar la función para exigir companyId o clabeAccountId obligatorio (validación dura).

• Reescribir las consultas para filtrar por tenant siempre (WHERE company_id=$X).

• Crear tests de autorización que validen que un usuario no puede listar transacciones de otra empresa.

8) getSavedAccountById consulta solo por ID (footgun de IDOR) [ALTO]

Ubicación: src/lib/db.ts (líneas 1650–1654)

Cómo ocurre / por qué existe: La función devuelve una cuenta guardada solo por su ID. Si un endpoint llama esto sin filtrar por user/company, puede exponer datos de terceros.

Evidencia (extracto):

```
1650: export async function getSavedAccountById(id: string): Promise<DbSavedAccount | null> {
1652:   SELECT * FROM saved_accounts WHERE id = ${id}
```

Impacto bancario: Filtración de cuentas destino/beneficiarios entre usuarios; soporte a fraude por selección de cuentas ajenas.

Instrucciones de corrección (paso a paso):

• Modificar la firma para incluir userId/companyId y filtrar en SQL.

• Revisar todas las llamadas a getSavedAccountById y reemplazarlas por getSavedAccountsByUserId cuando corresponda.

• Añadir constraint y/o políticas a nivel DB (por ejemplo, views o RLS si fuera Postgres con RLS).

9) Reset de contraseña con tokens en memoria y desactivación de 2FA [ALTO]

Ubicación: src/app/api/auth/reset-password/route.ts (líneas 10–11, 63, 152)

Cómo ocurre / por qué existe: El flujo guarda tokens en un Map en memoria (no compartido entre instancias) y además desactiva TOTP al completar el reset.

Evidencia (extracto):

```
0010: // Simple in-memory store for reset tokens ...
0011: const resetTokens = new Map<string, { email: string; expiresAt: number }>();
0063: resetTokens.set(code, { email, expiresAt });
0152: await disableTotp(user.id);
```

Impacto bancario: En EC2 con múltiples procesos/instancias, los resets fallan o se vuelven inconsistentes; desactivar 2FA reduce protección ante compromisos.

Instrucciones de corrección (paso a paso):

• Persistir tokens en DB/Redis (hasheados) con TTL y rate-limit por usuario/IP.

• Separar el concepto 'reset password' de 'deshabilitar 2FA' (si se permite, requerir verificación adicional).

• Registrar auditoría y notificar al usuario por canal out-of-band al ocurrir reset/2FA changes.

10) Operación privilegiada protegida solo con lock en memoria [ALTO]

Ubicación: src/app/api/orders/confirm-pending/route.ts (líneas 10–15, 57, 72–73, 225–226)

Cómo ocurre / por qué existe: El endpoint usa una variable booleana en memoria para evitar ejecución concurrente. En entornos con múltiples instancias, cada instancia tiene su propia memoria, por lo que el lock no es global.

Evidencia (extracto):

```
0010: // SECURITY FIX: In-memory lock to prevent concurrent execution
0012: let isProcessing = false;
```

```
0057: if (isProcessing) {
0072: // Acquire lock
0073: isProcessing = true;
0226: isProcessing = false;
```

Impacto bancario: Procesamiento duplicado o concurrente de confirmaciones/pending; riesgo operacional (dobles intentos) y estados inconsistentes.

Instrucciones de corrección (paso a paso):

• Mover a lock distribuido (Redis) o a cola (SQS FIFO) con worker único para este job.

• Restringir el endpoint a roles específicos (super_admin) y auditar cada ejecución.

• Asegurar idempotencia fuerte por transacción/orden antes de mutar estados.

11) Cliente OPM usa NEXT_PUBLIC para base URL y permite header auth vacío [MEDIO]

Ubicación: src/lib/opm-api.ts (líneas 14 y 94)

Cómo ocurre / por qué existe: API_BASE_URL se define con NEXT_PUBLIC (exponible en bundle) y el header X-Custom-Auth usa '' si no hay apiKey. Eso permite requests con auth faltante por mala configuración y expone detalles del proveedor.

Evidencia (extracto):

```
0014: const API_BASE_URL = process.env.NEXT_PUBLIC_OPM_API_URL || 'https://apiuat.opm.mx';
0094: 'X-Custom-Auth': apiKey || process.env.OPM_API_KEY || '',
```

Impacto bancario: Riesgo de configuración errónea y exposición innecesaria de endpoints/proveedor; degradación operativa ante auth faltante.

Instrucciones de corrección (paso a paso):

• Cambiar a variable server-only (OPM_API_URL) y evitar NEXT_PUBLIC.

• Abortar (throw) si no hay apiKey (no enviar requests con auth vacío).

• Centralizar retry/backoff solo para errores transitorios y usar idempotencia para operaciones sensibles.

12) Content-Security-Policy con valor truncado ('...') [MEDIO]

Ubicación: next.config.js (línea 31)

Cómo ocurre / por qué existe: La CSP contiene literalmente un '...' dentro del string: esto suele indicar un placeholder/truncamiento que vuelve la política inválida o no efectiva.

Evidencia (extracto):

```
0031: { key: 'Content-Security-Policy', value: "defaul...f' https:; worker-src 'self' blob: https://cdn.jsdeli
```

Impacto bancario: Sin CSP efectiva, cualquier XSS tendría mayor impacto (carga de scripts remotos, robo de sesión, acciones no autorizadas).

Instrucciones de corrección (paso a paso):

• Definir CSP completa y válida (sin '...'), alineada a los recursos reales del sitio.

• Preferir nonces/hashes para scripts, y añadir frame-ancestors 'none' (aunque también haya X-Frame-Options).

• Validar CSP con herramientas (browser devtools / csp-evaluator) en staging.

13) Endpoint de depuración que prueba endpoints no documentados del proveedor [MEDIO]

Ubicación: src/app/api/debug/probe-opm/route.ts (documentación en cabecera; habilitable por ENABLE_DEBUG_ENDPOINTS)

Cómo ocurre / por qué existe: Existe un endpoint diseñado para 'probar' rutas no documentadas. Aunque tiene un gate por env, su sola presencia incrementa superficie y riesgo de habilitación accidental.

Evidencia (extracto):

```
0005: POST /api/debug/probe-opm - Probe OPM API for undocumented endpoints
0011: WARNING: This endpoint should be removed or disabled in production
0107: Debug endpoint disabled in production. Set ENABLE_DEBUG_ENDPOINTS=true to enable.
```

Impacto bancario: Riesgo operacional por habilitación accidental; puede exponer comportamiento del proveedor y generar tráfico no deseado.

Instrucciones de corrección (paso a paso):

• Eliminar el endpoint de builds de producción (no solo bloquear por env).

• Si debe existir en staging: restringir por IP interna/VPN y por rol super_admin + auditoría obligatoria.

• Añadir un test de build que falle si el archivo existe en bundle prod.

Apéndice – Recomendaciones de despliegue seguro (AWS EC2)

• Colocar la app detrás de ALB; EC2 en subnets privadas; SG de EC2 solo permite tráfico desde SG del ALB.

• Asociar AWS WAF al ALB: IPSet allowlist para webhooks + rate-based rules para /api/auth y /api/webhooks.

• Mover secretos a AWS Secrets Manager; rotación; IAM least-privilege para EC2 (instance role).

• Centralizar logs en CloudWatch; métricas/alertas para fallos de firma webhooks, picos de 401/403 y resets de password.