🔒 INFORME DE AUDITORÍA DE SEGURIDAD - INMOVA

Fecha: 7 de Diciembre, 2025
Auditor: Arquitecto de Software Senior & Experto en Ciberseguridad
Aplicación: INMOVA - Sistema de Gestión de Propiedades
Versión: Next.js 15.5.7

📋 RESUMEN EJECUTIVO

✅ Fortalezas Detectadas

- ✓ Autenticación robusta con NextAuth.js y bcrypt
- ✓ Middleware con control de acceso basado en roles (RBAC)
- ✓ Rate limiting implementado con Redis/fallback in-memory
- ✓ CSP (Content Security Policy) estricto configurado
- ✓ Headers de seguridad HTTP implementados
- ✓ Uso de Prisma ORM (protección contra SQL Injection)
- ✓ Sentry configurado para monitoreo de errores

🚨 VULNERABILIDADES CRÍTICAS

🔴 CRÍTICO - A1: Broken Access Control

Ubicación: `/app/api/partners/login/route.ts`, `/app/api/partners/register/route.ts`

```
// ❌ PROBLEMA: Múltiples instancias de PrismaClient
const prisma = new PrismaClient();
```

Impacto:
- Agotamiento de conexiones de base de datos
- Posible DoS (Denial of Service)
- Violación del patrón Singleton

Solución:

```
// ✅ CORRECTO: Usar el singleton existente
import { prisma } from '@/lib/db';
```

Archivos afectados:
- `/app/api/partners/login/route.ts` (línea 8)
- `/app/api/partners/register/route.ts` (línea 7)

🔴 CRÍTICO - A2: Cryptographic Failures

Ubicación: `.env` (línea 1-31)

```
# ❌ PROBLEMA: Credenciales sensibles expuestas
DATABASE_URL='postgresql://
role_587683780:5kWw7vKJBDp9ZA2Jfkt5BdWrAjR0XDe5@db-587683780.db003.hosteddb.reai.io:
5432/587683780'
NEXTAUTH_SECRET=wJqizZO73C6pU4tjLTNwzjeoGLaMWvr9
ENCRYPTION_KEY=2ba4df6979746aca95db00f38050300c67601eb594c1db604dce7b3d5292e9c9
CRON_SECRET=fa787a4d35969abc72ce755a816d1031445e846ca4fa52983569d97143ea2210
```

Impacto:
- Acceso no autorizado a base de datos
- Compromiso de sesiones de usuario
- Violación de datos sensibles (GDPR)

Solución:
1. **INMEDIATO:** Rotar todas las credenciales comprometidas
2. **REQUERIDO:** Usar AWS Secrets Manager / HashiCorp Vault
3. **VERIFICAR:** `.env` debe estar en `.gitignore`

```
# ✅ CORRECTO: Usar gestión de secretos
# .env.example (solo plantillas)
DATABASE_URL=postgresql://user:password@host:5432/db
NEXTAUTH_SECRET=your_secret_here_32_chars_min
```

🟠 ALTO - A3: Injection (SQL Injection vía Prisma)

Ubicación: `/app/api/candidates/route.ts` (líneas 37-96)

```
// ⚠️ PROBLEMA: Falta validación de entrada
const body = await request.json();
const {
  unitId,
  nombreCompleto,
  dni,
  email,
  telefono,
  // ... sin validación
} = body;

// Directamente insertado en DB
const candidate = await prisma.candidate.create({
  data: {
    unitId,
    nombreCompleto,
    dni,
    email,
    // ...
  },
});
```

Impacto:
- Inyección de datos maliciosos
- XSS almacenado
- Corrupción de datos

Solución:

```
// ✅ CORRECTO: Validar con Zod
import { z } from 'zod';
import { sanitizeInput, sanitizeEmail } from '@/lib/security/sanitize';

const candidateSchema = z.object({
  unitId: z.string().uuid(),
  nombreCompleto: z.string().min(1).max(200),
  dni: z.string().regex(/^[0-9]{8}[A-Z]$/),
  email: z.string().email(),
  telefono: z.string().regex(/^[+0-9\s()-]{9,20}$/),
  fechaNacimiento: z.string().datetime(),
  situacionLaboral: z.enum(['empleado', 'autonomo', 'pensionista', 'estudiante', 'dese
mpleado']),
  ingresosMensuales: z.number().min(0).max(1000000),
  notas: z.string().max(5000).optional(),
});

export async function POST(request: Request) {
  try {
    const session = await getServerSession(authOptions);
    if (!session) {
      return NextResponse.json({ error: 'No autorizado' }, { status: 401 });
    }

    const body = await request.json();

    // Validar esquema
    const validatedData = candidateSchema.parse(body);

    // Sanitizar inputs
    const sanitizedData = {
      ...validatedData,
      nombreCompleto: sanitizeInput(validatedData.nombreCompleto, 200),
      email: sanitizeEmail(validatedData.email),
      notas: validatedData.notas ? sanitizeInput(validatedData.notas, 5000) : null,
    };

    // Verificar que unitId pertenece a la compañía del usuario
    const unit = await prisma.unit.findFirst({
      where: {
        id: sanitizedData.unitId,
        building: {
          companyId: (session.user as any).companyId,
        },
      },
    });

    if (!unit) {
      return NextResponse.json({ error: 'Unidad no encontrada' }, { status: 404 });
    }

    // Calcular scoring
    let scoring = 50;
    if (sanitizedData.ingresosMensuales) {
      const ratio = unit.rentaMensual / sanitizedData.ingresosMensuales;
      if (ratio < 0.3) scoring = 90;
      else if (ratio < 0.4) scoring = 70;
      else if (ratio < 0.5) scoring = 50;
      else scoring = 30;
    }

    const candidate = await prisma.candidate.create({
```

```
      data: {
        ...sanitizedData,
        fechaNacimiento: new Date(sanitizedData.fechaNacimiento),
        scoring,
      },
      include: {
        unit: {
          include: {
            building: true,
          },
        },
      },
    });

    return NextResponse.json(candidate, { status: 201 });
  } catch (error) {
    if (error instanceof z.ZodError) {
      return NextResponse.json(
        { error: 'Datos inválidos', details: error.errors },
        { status: 400 }
      );
    }
    logger.error('Error creating candidate:', error);
    return NextResponse.json(
      { error: 'Error al crear candidato' },
      { status: 500 }
    );
  }
}
```

🟠 ALTO - A4: Insecure Design - XSS Prevention

Ubicación: `/lib/security/sanitize.ts` (líneas 70-86)

```
// ❌ PROBLEMA: Sanitización de HTML inadecuada
export function sanitizeHtml(html: string): string {
  // Esta es una implementación básica. En producción, usar DOMPurify o similar
  let sanitized = html;

  // Remover script tags
  sanitized = sanitized.replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/
gi, '');

  // Remover event handlers
  sanitized = sanitized.replace(/on\w+\s*=\s*["'][^"']*["']/gi, '');

  // Remover javascript: en hrefs
  sanitized = sanitized.replace(/javascript:/gi, '');

  return sanitized;
}
```

Problemas:

- Regex bypass posible con payloads ofuscados
- No maneja `<img src=x onerror=alert(1)>`
- No protege contra `data:text/html;base64,...`
- No valida atributos style con `expression()`

Solución:

```
// ✅ CORRECTO: Instalar y usar isomorphic-dompurify
import DOMPurify from 'isomorphic-dompurify';

export function sanitizeHtml(html: string): string {
  return DOMPurify.sanitize(html, {
    ALLOWED_TAGS: ['b', 'i', 'em', 'strong', 'a', 'p', 'br', 'ul', 'ol', 'li'],
    ALLOWED_ATTR: ['href', 'target', 'rel'],
    ALLOWED_URI_REGEXP: /^(?:https?|mailto):/i,
  });
}
```

Dependencia ya instalada: ✅ `isomorphic-dompurify@2.33.0` (package.json línea 146)

🟠 ALTO - A5: Security Misconfiguration

Ubicación: `/lib/csp-strict.ts` (líneas 28-32)

```
// ⚠️ PROBLEMA: CSP demasiado permisivo
const cspDirectives = [
  "default-src 'self'",
  `script-src 'self' 'nonce-${nonce}' 'strict-dynamic' https://vercel.live https://
va.vercel-scripts.com`,
  `style-src 'self' 'nonce-${nonce}' 'unsafe-inline' https://
fonts.googleapis.com`, // ⚠️ unsafe-inline
  "img-src 'self' data: https: blob:", // ⚠️ Todos los HTTPS
  "connect-src 'self' https://*.vercel.app https://*.pusher.com wss://*.pusher.com
https://*.stripe.com https://api.openai.com",
];
```

Problemas:

1. `'unsafe-inline'` en style-src permite inyección de CSS
2. `img-src https:` permite todas las imágenes HTTPS (posible data exfiltration)
3. Wildcards demasiado amplios (`*.vercel.app`)

Solución:

```
// ✅ CORRECTO: CSP más restrictivo
const cspDirectives = [
  "default-src 'self'",
  `script-src 'self' 'nonce-${nonce}' 'strict-dynamic'`,
  `style-src 'self' 'nonce-${nonce}' https://fonts.googleapis.com`, // Remover unsafe-
inline
  `img-src 'self' data: https://${process.env.AWS_BUCKET_NAME}.s3.${pro-
cess.env.AWS_REGION}.amazonaws.com https://docs.aws.amazon.com/images/prescriptive-
guidance/latest/patterns/images/pattern-img/e0dd6928-4fe0-47ab-954f-9de5563349d8/
images/b42c7dd9-4a72-4998-bf88-195c8f90ed3e.png https://fonts.gstatic.com`, // Lista
blanca específica
  `font-src 'self' data: https://fonts.gstatic.com https://fonts.googleapis.com`,
  `connect-src 'self' https://inmova.app https://*.stripe.com https://api.openai.com`,
  "frame-ancestors 'none'",
  "base-uri 'self'",
  "form-action 'self'",
  "upgrade-insecure-requests",
  "block-all-mixed-content",
  "object-src 'none'",
  "media-src 'self' blob:",
  "worker-src 'self' blob:",
  "manifest-src 'self'",
];
```

🟡 MEDIO - A6: Vulnerable and Outdated Components

Ubicación: `package.json`

Dependencias con versiones potencialmente vulnerables:

```
"next": "15.5.7",           // ✅ OK (última versión estable)
"react": "18.2.0",          // ⚠️ Desactualizado (actual: 18.3.1)
"next-auth": "4.24.11",     // ⚠️ Versión antigua (considerar Auth.js v5)
"mapbox-gl": "1.13.3",      // 🔴 CRÍTICO: Versión muy antigua (actual: 3.x)
"prisma": "6.7.0",          // ✅ OK
"bcryptjs": "2.4.3",        // ✅ OK
```

Solución:

```
# Actualizar dependencias críticas
yarn upgrade react@18.3.1 react-dom@18.3.1
yarn upgrade mapbox-gl@latest

# Revisar breaking changes de Auth.js v5
# https://authjs.dev/guides/upgrade-to-v5
```

🟡 MEDIO - A7: Identification and Authentication Failures

Ubicación: `/lib/auth-options.ts` (líneas 16-88)

Problemas:

1. **Falta implementación de rate limiting en login**

```
// ❌ PROBLEMA: Sin protección contra brute force
export async function POST(request: NextRequest) {
  const { email, password } = await request.json();
  // ... autenticación directa
}
```

1. **No hay protección contra timing attacks**

```
// ⚠️ PROBLEMA: bcrypt.compare puede revelar si el usuario existe
const user = await prisma.user.findUnique({ where: { email } });
if (!user) {
  throw new Error('Usuario no encontrado'); // ❌ Revela existencia
}
const isValid = await bcrypt.compare(password, user.password);
if (!isValid) {
  throw new Error('Contraseña incorrecta'); // ❌ Revela qué falló
}
```

1. **No hay sistema de account lockout**

2. **No hay logs de intentos fallidos**

Solución:

```
import { rateLimiters } from '@/lib/rate-limit-enhanced';
import { logSecurityEvent } from '@/lib/logger';

export async function POST(request: NextRequest) {
  // 1. Rate limiting por IP
  const identifier = getRateLimitIdentifier(request as any);
  const rateLimitResult = await rateLimiters.auth.checkLimit(identifier);

  if (!rateLimitResult.success) {
    return NextResponse.json(
      { error: 'Demasiados intentos. Intente más tarde.' },
      { status: 429, headers: applyRateLimitHeaders(new Headers(), rateLimitResult) }
    );
  }

  const { email, password } = await request.json();

  // 2. Validar formato
  if (!email || !password || !isValidEmail(email)) {
    return NextResponse.json({ error: 'Credenciales inválidas' }, { status: 401 });
  }

  // 3. Siempre hacer hash comparison para prevenir timing attacks
  const user = await prisma.user.findUnique({
    where: { email: email.toLowerCase() },
    include: { company: true },
  });

  const dummyHash = '$2a$10$YourDummyHashForTimingAttackPrevention';
  const hashToCompare = user?.password || dummyHash;
  const isPasswordValid = await bcrypt.compare(password, hashToCompare);

  // 4. Respuesta genérica (no revelar si el usuario existe)
  if (!user || !isPasswordValid || !user.activo) {
    // Log intento fallido
    logSecurityEvent('login_failed', email, {
      ip: identifier,
      reason: !user ? 'user_not_found' : !isPasswordValid ? 'invalid_password' : 'user
_inactive',
    });

    // Delay artificial (honeypot)
    await new Promise(resolve => setTimeout(resolve, Math.random() * 1000 + 500));

    return NextResponse.json(
      { error: 'Credenciales inválidas' }, // ✅ Mensaje genérico
      { status: 401 }
    );
  }

  // 5. Log login exitoso
  logSecurityEvent('login_success', user.id, { ip: identifier });

  // 6. Actualizar último acceso
  await prisma.user.update({
    where: { id: user.id },
    data: { lastLogin: new Date() },
  });

  return NextResponse.json({ success: true, user });
}
```

🟡 MEDIO - A8: Software and Data Integrity Failures

Ubicación: `/next.config.js`

```
// ❌ PROBLEMA: No hay Subresource Integrity (SRI)
const nextConfig = {
  // ...
  images: { unoptimized: true }, // ⚠️ Deshabilitado
};
```

Solución:

```
const nextConfig = {
  distDir: process.env.NEXT_DIST_DIR || '.next',
  output: process.env.NEXT_OUTPUT_MODE,
  experimental: {
    outputFileTracingRoot: path.join(__dirname, '../'),
  },
  eslint: {
    ignoreDuringBuilds: false, // ✅ Habilitar ESLint en builds
  },
  typescript: {
    ignoreBuildErrors: false, // ✅ Ya correcto
  },
  images: {
    unoptimized: false, // ✅ Habilitar optimización
    remotePatterns: [
      {
        protocol: 'https',
        hostname: `${process.env.AWS_BUCKET_NAME}.s3.${process.env.AWS_REGION}.amazon-
aws.com`,
      },
      {
        protocol: 'https',
        hostname: 'abacusai-apps*.s3.*.amazonaws.com',
      },
    ],
  },
  // SRI para scripts externos
  webpack: (config) => {
    config.module.rules.push({
      test: /\.html$/,
      use: 'html-loader',
    });
    return config;
  },
};
```

🟡 MEDIO - A9: Security Logging and Monitoring Failures

Ubicación: `/lib/logger.ts`

Problemas:

1. Logger muy básico (solo console)
2. No hay correlación de logs (request ID)

3. No hay alertas automáticas para eventos críticos
4. Logs no estructurados para análisis

Solución:

```typescript
import { randomUUID } from 'crypto';
import winston from 'winston';

// ✅ CORRECTO: Winston con formato JSON
const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  defaultMeta: {
    service: 'inmova-api',
    environment: process.env.NODE_ENV,
  },
  transports: [
    new winston.transports.File({
      filename: 'logs/error.log',
      level: 'error',
      maxsize: 10485760, // 10MB
      maxFiles: 5,
    }),
    new winston.transports.File({
      filename: 'logs/combined.log',
      maxsize: 10485760,
      maxFiles: 10,
    }),
  ],
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.combine(
      winston.format.colorize(),
      winston.format.simple()
    ),
  }));
}

// Context con request ID
export function createRequestLogger(requestId?: string) {
  const id = requestId || randomUUID();
  return logger.child({ requestId: id });
}

// Helper para eventos de seguridad críticos
export const logSecurityEvent = async (
  event: string,
  userId: string,
  details: Record<string, any>
) => {
  logger.warn('Security Event', {
    event,
    userId,
    timestamp: new Date().toISOString(),
    ...details,
  });

  // Alerta para eventos críticos
  const criticalEvents = [
    'multiple_failed_logins',
    'privilege_escalation_attempt',
```

```
    'sql_injection_attempt',
    'xss_attempt',
  ];

  if (criticalEvents.includes(event)) {
    // Enviar a Sentry o sistema de alertas
    await sendSecurityAlert(event, userId, details);
  }
};

export default logger;
```

🟢 BAJO - A10: Server-Side Request Forgery (SSRF)

Ubicación: `/lib/security/sanitize.ts` (líneas 21-33)

Problema:

```
export function sanitizeUrl(url: string): string {
  try {
    const parsed = new URL(url);
    // Solo permitir protocolos seguros
    if (!['http:', 'https:', 'mailto:'].includes(parsed.protocol)) {
      return '';
    }
    return parsed.href;
  } catch {
    return '';
  }
}
```

Falta validación de:
- IPs privadas (192.168.x.x, 10.x.x.x, 127.0.0.1)
- Localhost
- Metadata endpoints (169.254.169.254)

Solución:

```typescript
export function sanitizeUrl(url: string): string {
  try {
    const parsed = new URL(url);

    // Validar protocolo
    if (!['http:', 'https:', 'mailto:'].includes(parsed.protocol)) {
      return '';
    }

    // Prevenir SSRF a IPs privadas
    const hostname = parsed.hostname.toLowerCase();
    const privateIpPatterns = [
      /^127\./,
      /^10\./,
      /^172\.(1[6-9]|2[0-9]|3[01])\./,
      /^192\.168\./,
      /^169\.254\./, // AWS metadata
      /^localhost$/i,
      /^::1$/,
      /^0\.0\.0\.0$/,
    ];

    if (privateIpPatterns.some(pattern => pattern.test(hostname))) {
      logger.warn('SSRF attempt detected', { url, hostname });
      return '';
    }

    return parsed.href;
  } catch {
    return '';
  }
}
```

🛡️ RECOMENDACIONES ADICIONALES

1. Implementar Helmet.js (Security Headers)

```
yarn add helmet
```

```typescript
// middleware.ts
import helmet from 'helmet';

export default helmet({
  contentSecurityPolicy: false, // Ya tienes CSP custom
  hsts: {
    maxAge: 31536000,
    includeSubDomains: true,
    preload: true,
  },
  referrerPolicy: { policy: 'strict-origin-when-cross-origin' },
  frameguard: { action: 'deny' },
});
```

2. Implementar WAF (Web Application Firewall)

Si estás usando AWS:
- AWS WAF con reglas OWASP Top 10
- Cloudflare WAF (alternativa)

3. Penetration Testing Regular

Herramientas recomendadas:
- OWASP ZAP
- Burp Suite
- Nuclei

```
# Escaneo básico con OWASP ZAP
docker run -t owasp/zap2docker-stable zap-baseline.py \
  -t https://inmova.app \
  -r zap-report.html
```

4. Dependency Scanning

```
# Audit de dependencias
yarn audit --level high

# Snyk scan
npx snyk test

# OWASP Dependency Check
dependency-check --project INMOVA --scan .
```

5. Secrets Scanning

```
# Gitleaks para detectar secretos en commits
gitleaks detect --source . --verbose

# TruffleHog
trufflehog filesystem . --json
```

📊 MATRIZ DE RIESGOS

Vulnerabilidad	Severidad	Impacto	Probabilidad	Prioridad
Múltiples PrismaClient	CRÍTICO	Alto	Alto	P0 - Inmediato
Credenciales expuestas	CRÍTICO	Crítico	Medio	P0 - Inmediato
Falta validación inputs	ALTO	Alto	Alto	P1 - 1 semana
Sanitización HTML débil	ALTO	Medio	Medio	P1 - 1 semana
CSP permisivo	ALTO	Medio	Bajo	P2 - 2 semanas
Dependencias antiguas	MEDIO	Medio	Alto	P2 - 2 semanas
Auth timing attacks	MEDIO	Medio	Bajo	P3 - 1 mes
Logging insuficiente	MEDIO	Bajo	Medio	P3 - 1 mes
SSRF no validado	BAJO	Bajo	Muy Bajo	P4 - Backlog

✅ CHECKLIST DE REMEDIACIÓN

Semana 1 (Crítico)

- [ ] Reemplazar `new PrismaClient()` por `prisma` singleton
- [ ] Rotar todas las credenciales expuestas en `.env`
- [ ] Implementar AWS Secrets Manager
- [ ] Verificar `.gitignore` incluye `.env`
- [ ] Ejecutar `yarn audit` y resolver vulnerabilidades críticas

Semana 2 (Alto)

- [ ] Implementar validación Zod en todos los endpoints POST/PUT
- [ ] Implementar DOMPurify para sanitización HTML
- [ ] Restringir CSP (remover `unsafe-inline`, especificar dominios)
- [ ] Actualizar React, Mapbox y otras dependencias críticas

Semana 3-4 (Medio)

- [ ] Implementar rate limiting en endpoints de autenticación
- [ ] Mejorar logger con Winston (logs estructurados)
- [ ] Implementar sistema de alertas para eventos críticos
- [ ] Agregar validación anti-SSRF en sanitizeUrl
- [ ] Implementar account lockout tras 5 intentos fallidos

Mes 2 (Mejora Continua)

- [ ] Configurar Snyk o Dependabot para escaneo automático
- [ ] Implementar Helmet.js para headers adicionales
- [ ] Configurar AWS WAF con reglas OWASP
- [ ] Establecer proceso de Penetration Testing trimestral
- [ ] Capacitación del equipo en OWASP Top 10

📞 CONTACTO

Auditor: Arquitecto de Software Senior
Fecha: 7 de Diciembre, 2025
Próxima revisión: Marzo 2026 (trimestral)

Firma Digital: ✅ Auditoria completada satisfactoriamente