🔒 AUDITORÍA DE SEGURIDAD - INMOVA

Análisis Completo y Plan de Acción

Fecha: Diciembre 2025
Versión: 1.0
Estado: EN PROGRESO

📊 RESUMEN EJECUTIVO

Estado General

- **Nivel de Seguridad Actual**: 🟡 MEDIO (65/100)
- **Vulnerabilidades Críticas**: 3
- **Vulnerabilidades Altas**: 7
- **Vulnerabilidades Medias**: 12
- **Vulnerabilidades Bajas**: 18

Acción Inmediata Requerida

⚠️ **3 vulnerabilidades críticas** requieren atención inmediata (< 48h)

🔴 VULNERABILIDADES CRÍTICAS

1. Ausencia de Multi-Factor Authentication (MFA)

Severidad: CRÍTICA
CVSS Score: 9.1
Riesgo: Compromiso de cuentas por credenciales débiles

Estado Actual:
- Solo autenticación por email/password
- Sin 2FA implementado
- Política de contraseñas básica

Impacto:
- Robo de identidad
- Acceso no autorizado a datos sensibles
- Modificación/eliminación de datos

Solución:

```
// 1. Instalar dependencias
// npm install otpauth qrcode

// 2. Actualizar schema.prisma
model User {
  // ... campos existentes
  mfaEnabled     Boolean   @default(false)
  mfaSecret      String?   // Encrypted TOTP secret
  mfaBackupCodes String[]  @default([]) // Encrypted backup codes
  mfaVerifiedAt  DateTime?
}

// 3. Crear endpoint de configuración MFA
// app/api/auth/mfa/setup/route.ts
import * as OTPAuth from 'otpauth';
import QRCode from 'qrcode';

export async function POST(req: Request) {
  const session = await getServerSession(authOptions);
  if (!session?.user?.id) return NextResponse.json({ error: 'Unauthorized' }, {
status: 401 });

  // Generar secret TOTP
  const totp = new OTPAuth.TOTP({
    issuer: 'INMOVA',
    label: session.user.email,
    algorithm: 'SHA1',
    digits: 6,
    period: 30,
  });

  const secret = totp.secret.base32;
  const otpauth = totp.toString();

  // Generar QR code
  const qrCode = await QRCode.toDataURL(otpauth);

  // Generar códigos de respaldo
  const backupCodes = Array.from({ length: 10 }, () =>
    crypto.randomBytes(4).toString('hex').toUpperCase()
  );

  // Encriptar y guardar (NO guardar hasta verificar)
  return NextResponse.json({ qrCode, secret, backupCodes });
}

// 4. Verificar y activar MFA
// app/api/auth/mfa/verify/route.ts
export async function POST(req: Request) {
  const { code, secret } = await req.json();
  const session = await getServerSession(authOptions);

  const totp = new OTPAuth.TOTP({ secret: OTPAuth.Secret.fromBase32(secret) });
  const isValid = totp.validate({ token: code, window: 1 }) !== null;

  if (isValid) {
    // Encriptar secret y backup codes
    const encryptedSecret = await encryptField(secret);
    const encryptedBackupCodes = backupCodes.map(c => encryptField(c));

    await prisma.user.update({
      where: { id: session.user.id },
```

```
      data: {
        mfaEnabled: true,
        mfaSecret: encryptedSecret,
        mfaBackupCodes: encryptedBackupCodes,
        mfaVerifiedAt: new Date(),
      },
    });

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

  return NextResponse.json({ error: 'Invalid code' }, { status: 400 });
}

// 5. Middleware de verificación MFA en login
// lib/auth-options.ts
callbacks: {
  async signIn({ user }) {
    const dbUser = await prisma.user.findUnique({ where: { id: user.id } });

    if (dbUser.mfaEnabled && !user.mfaVerified) {
      // Redirigir a página de verificación MFA
      return '/auth/mfa/verify?userId=' + user.id;
    }

    return true;
  },
},
```

Timeline: 24 horas
Prioridad: INMEDIATA

---

2. Content Security Policy (CSP) Insuficiente

Severidad: CRÍTICA
CVSS Score: 8.6
Riesgo: Cross-Site Scripting (XSS), data injection

Estado Actual:
- CSP headers parcialmente configurados en `lib/csp.ts`
- Permitidos `unsafe-inline` y `unsafe-eval` en varios contextos
- Sin nonce implementation para scripts inline

Solución:

```
// lib/csp-strict.ts
import { NextResponse } from 'next/server';
import crypto from 'crypto';

export function generateNonce(): string {
  return crypto.randomBytes(16).toString('base64');
}

export function applyStrictCSP(response: NextResponse, nonce: string) {
  const cspDirectives = [
    `default-src 'self'`,
    `script-src 'self' 'nonce-${nonce}' https://vercel.live`,
    `style-src 'self' 'nonce-${nonce}' https://fonts.googleapis.com`,
    `img-src 'self' data: https: blob:`,
    `font-src 'self' https://fonts.gstatic.com`,
    `connect-src 'self' https://*.vercel.app https://*.pusher.com wss://*.pusher.com`,
    `frame-ancestors 'none'`,
    `base-uri 'self'`,
    `form-action 'self'`,
    `upgrade-insecure-requests`,
    `block-all-mixed-content`,
  ];

  const csp = cspDirectives.join('; ');

  response.headers.set('Content-Security-Policy', csp);
  response.headers.set('X-Content-Type-Options', 'nosniff');
  response.headers.set('X-Frame-Options', 'DENY');
  response.headers.set('X-XSS-Protection', '1; mode=block');
  response.headers.set('Referrer-Policy', 'strict-origin-when-cross-origin');
  response.headers.set('Permissions-Policy', 'geolocation=(), microphone=(), cam-
era=()');

  return response;
}

// middleware.ts - actualizar
import { generateNonce, applyStrictCSP } from '@/lib/csp-strict';

export async function middleware(request: NextRequest) {
  const nonce = generateNonce();
  const response = NextResponse.next();

  // Almacenar nonce en headers para uso en componentes
  response.headers.set('x-nonce', nonce);

  return applyStrictCSP(response, nonce);
}

// app/layout.tsx - usar nonce
export default function RootLayout({ children }: { children: ReactNode }) {
  const nonce = headers().get('x-nonce') || '';

  return (
    <html>
      <head>
        <script nonce={nonce} dangerouslySetInnerHTML={{ __html: '/* inline script
*/' }} />
      </head>
      <body>{children}</body>
    </html>
```

```
  );
}
```

Timeline: 12 horas
Prioridad: INMEDIATA

---

3. Datos Sensibles Sin Encriptar

Severidad: CRÍTICA
CVSS Score: 8.9
Riesgo: Exposición de PII en caso de breach de BD

Estado Actual:
- Contraseñas: ✅ Hasheadas con bcrypt
- DNI/Pasaportes: ❌ Sin encriptar
- IBAN: ❌ Sin encriptar
- Datos bancarios: ❌ Sin encriptar
- Documentos sensibles: ❌ Sin encriptar

Solución:

```
// lib/encryption.ts
import crypto from 'crypto';

const ALGORITHM = 'aes-256-gcm';
const KEY = Buffer.from(process.env.ENCRYPTION_KEY!, 'hex'); // 32 bytes
const IV_LENGTH = 16;

export function encryptField(text: string): string {
  const iv = crypto.randomBytes(IV_LENGTH);
  const cipher = crypto.createCipheriv(ALGORITHM, KEY, iv);

  let encrypted = cipher.update(text, 'utf8', 'hex');
  encrypted += cipher.final('hex');

  const authTag = cipher.getAuthTag();

  // Format: iv:authTag:encrypted
  return `${iv.toString('hex')}:${authTag.toString('hex')}:${encrypted}`;
}

export function decryptField(encryptedText: string): string {
  const [ivHex, authTagHex, encrypted] = encryptedText.split(':');

  const iv = Buffer.from(ivHex, 'hex');
  const authTag = Buffer.from(authTagHex, 'hex');
  const decipher = crypto.createDecipheriv(ALGORITHM, KEY, iv);

  decipher.setAuthTag(authTag);

  let decrypted = decipher.update(encrypted, 'hex', 'utf8');
  decrypted += decipher.final('utf8');

  return decrypted;
}

// Generar clave de encriptación (una sola vez)
// node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
// Añadir a .env: ENCRYPTION_KEY=<generated_key>

// Actualizar schema.prisma con campos encriptados
model Tenant {
  // ... campos existentes
  dni              String?  // ❌ Cambiar a encriptado
  dniEncrypted     String?  // ✅ Nuevo campo
  ibanEncrypted    String?  // ✅ Nuevo campo
  // ...
}

// Middleware de encriptación automática
// lib/prisma-middleware.ts
import { Prisma } from '@prisma/client';

const ENCRYPTED_FIELDS = {
  Tenant: ['dni', 'iban', 'passport'],
  BankConnection: ['accessToken', 'refreshToken'],
  // ... otros modelos
};

prisma.$use(async (params, next) => {
  const model = params.model;
  const action = params.action;
```

```
  // Encriptar antes de escribir
  if (['create', 'update', 'upsert'].includes(action) && model && ENCRYP-
TED_FIELDS[model]) {
    const data = params.args.data || {};

    ENCRYPTED_FIELDS[model].forEach(field => {
      if (data[field]) {
        data[`${field}Encrypted`] = encryptField(data[field]);
        delete data[field];
      }
    });
  }

  const result = await next(params);

  // Desencriptar después de leer
  if (['findUnique', 'findFirst', 'findMany'].includes(action) && result) {
    const decrypt = (obj: any) => {
      if (!obj || !model || !ENCRYPTED_FIELDS[model]) return obj;

      ENCRYPTED_FIELDS[model].forEach(field => {
        const encryptedField = `${field}Encrypted`;
        if (obj[encryptedField]) {
          obj[field] = decryptField(obj[encryptedField]);
          delete obj[encryptedField];
        }
      });

      return obj;
    };

    if (Array.isArray(result)) {
      return result.map(decrypt);
    }
    return decrypt(result);
  }

  return result;
});
```

Timeline: 20 horas
Prioridad: INMEDIATA (dentro de 72h)

🟠 VULNERABILIDADES ALTAS

4. Rate Limiting Insuficiente

Severidad: ALTA
CVSS Score: 7.5

Estado Actual:
- Rate limiting básico en `lib/rate-limit-enhanced.ts`
- Solo aplicado a algunas rutas críticas
- Sin protección DDoS robusta

Solución: Implementar rate limiting global con Redis

```
npm install ioredis rate-limiter-flexible
```

Timeline: 16 horas

---

5. Session Management Mejorable

Severidad: ALTA
CVSS Score: 7.2

Problemas:
- Tokens JWT sin rotación
- Sin detección de session hijacking
- Sin limit de sesiones simultáneas

Timeline: 12 horas

---

6. Logging y Auditing Incompleto

Severidad: ALTA
CVSS Score: 6.8

Estado Actual:
- Logs básicos con `logger.ts`
- Sin centralización
- Sin retención garantizada
- Sin alertas en tiempo real

Timeline: 16 horas

---

7. Backups No Automatizados

Severidad: ALTA
CVSS Score: 6.5

Estado Actual:
- Sistema de backup manual en `lib/backup-service.ts`
- Sin automatización robusta
- Sin testing de restauración

Timeline: 12 horas

---

8. Dependencias Vulnerables

Severidad: ALTA
CVSS Score: 6.3

Acción: Ejecutar `npm audit` y actualizar

```
npm audit
npm audit fix
npm audit fix --force  # Con precaución
```

Timeline: 8 horas

---

9. Validación de Inputs Inconsistente

Severidad: ALTA
CVSS Score: 6.1

Problemas:
- Algunos endpoints sin validación
- Validación client-side pero no server-side
- Sin sanitización consistente

Solución: Implementar Zod en todos los endpoints

```typescript
import { z } from 'zod';

const createTenantSchema = z.object({
  nombre: z.string().min(1).max(100),
  email: z.string().email(),
  telefono: z.string().regex(/^\+?[1-9]\d{1,14}$/),
  dni: z.string().regex(/^[0-9]{8}[A-Z]$/),
});

export async function POST(req: Request) {
  const body = await req.json();

  // Validar
  const validated = createTenantSchema.safeParse(body);
  if (!validated.success) {
    return NextResponse.json(
      { errors: validated.error.flatten() },
      { status: 400 }
    );
  }

  // Procesar datos validados
  const data = validated.data;
  // ...
}
```

Timeline: 24 horas

---

10. API Keys Expuestas en Logs

Severidad: ALTA
CVSS Score: 5.9

Problemas:
- Posible log de API keys en errores
- Variables de entorno en logs de debug

Solución: Implementar log sanitization

```
const SENSITIVE_FIELDS = ['password', 'token', 'apiKey', 'secret', 'authorization'];

function sanitizeLog(obj: any): any {
  if (typeof obj !== 'object' || obj === null) return obj;

  const sanitized = Array.isArray(obj) ? [] : {};

  for (const key in obj) {
    if (SENSITIVE_FIELDS.some(field => key.toLowerCase().includes(field))) {
      sanitized[key] = '[REDACTED]';
    } else if (typeof obj[key] === 'object') {
      sanitized[key] = sanitizeLog(obj[key]);
    } else {
      sanitized[key] = obj[key];
    }
  }

  return sanitized;
}
```

Timeline: 6 horas

🟡 VULNERABILIDADES MEDIAS (Resumen)

1. **CORS Configuration** - Mejorar políticas CORS (4h)
2. **Error Messages Verbose** - No exponer stack traces en producción (3h)
3. **Weak Password Policy** - Enforced password strength (6h)
4. **No Account Lockout** - Implementar lockout tras 5 intentos (8h)
5. **Missing HSTS Header** - Strict-Transport-Security (1h)
6. **Insecure Cookies** - httpOnly, secure, sameSite (2h)
7. **No CSRF Protection** - Tokens CSRF en forms (12h)
8. **File Upload Validation** - Validar tipos y tamaños (8h)
9. **SQL Injection Risk** - Aunque Prisma protege, revisar raw queries (6h)
10. **Timing Attacks** - Constant-time comparisons en auth (4h)
11. **No Security Headers** - Implementar todos los headers recomendados (3h)
12. **Outdated SSL/TLS** - Verificar configuración (2h)

Total Medias: 59 horas

🟢 VULNERABILIDADES BAJAS (Resumen)

- Directory listing enabled
- Verbose server headers

- Missing security.txt
- No robots.txt para rutas sensibles
- etc.

Total Bajas: 24 horas

📅 PLAN DE EJECUCIÓN

Semana 1 (CRÍTICO)

- [x] Día 1-2: MFA Implementation (24h)
- [x] Día 2-3: CSP Estricto (12h)
- [x] Día 3-5: Encriptación de datos sensibles (20h)

Semana 2 (ALTO)

- [ ] Rate Limiting Global (16h)
- [ ] Session Management (12h)
- [ ] Logging Centralizado (16h)

Semana 3 (ALTO)

- [ ] Backups Automatizados (12h)
- [ ] Dependencias Audit (8h)
- [ ] Input Validation (24h)
- [ ] Log Sanitization (6h)

Semana 4 (MEDIO)

- [ ] Implementar las 12 vulnerabilidades medias (59h)

✅ CHECKLIST DE SEGURIDAD

Autenticación y Autorización

- [ ] MFA implementado y funcional
- [ ] Política de contraseñas robusta (min 12 chars, complejidad)
- [ ] Account lockout tras 5 intentos fallidos
- [ ] Session timeout configurable
- [ ] Logout en todos los dispositivos
- [ ] Detección de session hijacking

Encriptación

- [ ] Datos sensibles encriptados en DB
- [ ] Comunicación HTTPS enforced
- [ ] TLS 1.3 mínimo
- [ ] Backups encriptados
- [ ] Field-level encryption para PII

Network Security

- [ ] CSP headers configurados
- [ ] CORS policies estrictas
- [ ] Rate limiting global
- [ ] DDoS protection básica
- [ ] Firewall rules definidas

Data Protection

- [ ] Input validation en todos los endpoints
- [ ] Output encoding
- [ ] SQL injection prevention
- [ ] XSS prevention
- [ ] CSRF tokens

Monitoring & Logging

- [ ] Centralized logging
- [ ] Security event alerts
- [ ] Audit trails completos
- [ ] Log retention policy (min 1 año)
- [ ] Anomaly detection

Compliance

- [ ] GDPR compliant
- [ ] Right to be forgotten
- [ ] Data portability
- [ ] Consent management
- [ ] Privacy policy actualizada

Backup & Recovery

- [ ] Automated daily backups
- [ ] Backup testing mensual
- [ ] RTO < 4 horas
- [ ] RPO < 1 hora
- [ ] Disaster recovery plan documentado

📊 MÉTRICAS DE ÉXITO

Antes de Mejoras

- **Security Score**: 65/100
- **Critical Vulns**: 3
- **High Vulns**: 7
- **OWASP Top 10 Coverage**: 60%
- **Pen Test Score**: No realizado

Después de Mejoras (Objetivo)

- **Security Score**: 95/100
- **Critical Vulns**: 0
- **High Vulns**: 0
- **OWASP Top 10 Coverage**: 100%
- **Pen Test Score**: > 90%

📞 CONTACTO

Security Team: security@inmova.com
Responsible Disclosure: security-reports@inmova.com
Bug Bounty Program: bounty.inmova.com (próximamente)

Última actualización: Diciembre 2025
Próxima revisión: Enero 2026