Informe de Análisis de Seguridad - Aplicación Médica

Fecha de análisis: 2025-01-27
Versión analizada: Mayor Update (post-implementación de mejoras de seguridad)
Estándar de referencia: OWASP Application Security Verification Standard (ASVS)

---

1. Descripción Breve de la Aplicación Analizada

1.1. Propósito y Funcionalidad

La aplicación es una herramienta web **monousuario** diseñada para el registro personal de peso, talla y cálculo del Índice de Masa Corporal (IMC). Su objetivo principal es permitir a un único usuario realizar un seguimiento de su peso corporal y obtener información sobre su estado nutricional mediante el cálculo automático del IMC.

1.2. Características Principales

- **Registro de datos personales**: Nombre, apellidos, fecha de nacimiento y talla (en metros)
- **Registro de peso**: Permite registrar el peso actual con fecha y hora automáticas
- **Cálculo automático de IMC**: Calcula y muestra el Índice de Masa Corporal basado en el último peso registrado
- **Estadísticas históricas**: Muestra número de pesajes, peso máximo y peso mínimo registrados
- **Sincronización bidireccional**: Entre frontend (localStorage) y backend (memoria)
- **Modo offline**: Funciona sin conexión al servidor utilizando almacenamiento local

1.3. Arquitectura Técnica

- **Backend**: Flask (Python) con API REST
- **Frontend**: JavaScript vanilla con almacenamiento en localStorage
- **Almacenamiento**: Memoria en backend + localStorage en frontend
- **Tests**: 86 tests backend (pytest) + ~66 tests frontend (Jest)
- **Gestión de vulnerabilidades**: DefectDojo integrado para seguimiento de debilidades de seguridad

1.4. Contexto de Seguridad

La aplicación maneja **datos de salud personales** (peso, altura, fecha de nacimiento), aunque en un contexto monousuario y sin autenticación compleja. No almacena datos médicos críticos ni información de identificación sensible, pero sí información personal que requiere protección adecuada.

---

2. Identificación de Vectores de Ataque Potenciales

Basado en el análisis CWE-699 (Software Development) realizado, se han identificado los siguientes vectores de ataque potenciales:

2.1. Vectores de Ataque Identificados

2.1.1. Inyección de Datos Maliciosos (Type Confusion)

CWE-1287: Improper Validation of Specified Type of Input
CWE-843: Access of Resource Using Incompatible Type

Ubicación: - Backend: `app/routes.py` (conversiones de `float()`) - Frontend: `app/static/js/main.js`, `app/static/js/storage.js` (conversiones de `parseFloat()`)

Vector de ataque: - Envío de valores no numéricos que se convierten a `NaN` o `Infinity` - Propagación de valores inválidos en cálculos matemáticos (IMC) - Causa de errores en la aplicación o resultados incorrectos

Ejemplo de ataque:

```
// Atacante envía: { "talla_m": "NaN" }
// Resultado: parseFloat("NaN") = NaN
// IMC calculado con NaN = NaN (resultado inválido)
```

2.1.2. Clickjacking (UI Redressing)

CWE-1021: Improper Restriction of Rendered UI Layers or Frames

Ubicación: `app/__init__.py` (falta de headers de seguridad)

Vector de ataque: - Inclusión de la aplicación en un iframe malicioso - Superposición de elementos invisibles sobre la interfaz - Manipulación de acciones del usuario sin su conocimiento

Ejemplo de ataque:

```
<!-- Sitio malicioso -->
<iframe src="http://aplicacion-medica.com" style="opacity:0.1"></iframe>
<button style="position:absolute; top:100px; left:100px">Click aquí</button>
<!-- Usuario cree que hace clic en el botón, pero realmente interactúa con la app -->
```

2.1.3. Cross-Origin Resource Sharing (CORS) Permisivo

CWE-942: Overly Permissive Cross-domain Whitelist

Ubicación: `app/__init__.py` (configuración CORS con `origins: "*"`)

Vector de ataque: - Cualquier sitio web puede realizar peticiones a la API - Aumento de la superficie de ataque si el backend se expone fuera de `localhost` - Posibilidad de realizar ataques CSRF desde dominios externos

Ejemplo de ataque:

```
// Desde cualquier sitio web malicioso
fetch('http://aplicacion-medica.com/api/user', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({nombre: 'Ataque', talla_m: 'NaN'})
});
```

2.1.4. Manejo Inadecuado de Excepciones

CWE-703: Improper Check or Handling of Exceptional Conditions

Ubicación: `app/routes.py` (uso de `Exception` genérico)

Vector de ataque: - Ocultación de errores inesperados que podrían indicar vulnerabilidades - Dificultad para detectar y responder a ataques - Información de error insuficiente para debugging y monitoreo

2.1.5. Validación de Entrada Insuficiente (Resuelto)

CWE-20: Improper Input Validation

Estado: ✅ **RESUELTO** - Se implementó validación robusta de nombres con sanitización

Ubicación: `app/helpers.py`, `app/routes.py`, `app/static/js/main.js`

Vector de ataque original: - Inyección de caracteres peligrosos (`< > " '`) - Ataques XSS potenciales - Corrupción de datos almacenados

---

3. Análisis de Riesgos Asociados

3.1. Matriz de Riesgos

CWE	Vector de Ataque	Probabilidad	Impacto	Riesgo	Estado
CWE-1287	Type Confusion (Backend)	Media	Medio	**MEDIO**	⚠️ Pendiente
CWE-843	Type Confusion (Frontend)	Media	Medio	**MEDIO**	⚠️ Pendiente
CWE-1021	Clickjacking	Baja	Medio	**MEDIO**	⚠️ Pendiente
CWE-942	CORS Permisivo	Baja*	Alto*	**MEDIO/ ALTO***	⏸️ Pendiente
CWE-703	Manejo de Excepciones	Baja	Bajo	**BAJO**	⚠️ Mejorado parcialmente
CWE-20	Validación de Entrada	Media	Medio	**MEDIO**	✅ Resuelto

**Riesgo aumenta significativamente si el backend se expone fuera de* `localhost`

3.2. Análisis Detallado de Riesgos

3.2.1. Riesgo MEDIO: Type Confusion (CWE-1287, CWE-843)

Descripción del riesgo: - Los valores `NaN` e `Infinity` pueden propagarse en cálculos matemáticos - Resultados de IMC incorrectos o inválidos - Posible corrupción de datos almacenados

Impacto en el negocio: - **Integridad de datos**: Los datos almacenados pueden volverse inválidos - **Confiabilidad**: Los usuarios pueden recibir información incorrecta sobre su IMC - **Experiencia de usuario**: Errores en la aplicación que afectan la usabilidad

Mitigación actual: - ✅ Validaciones defensivas antes de calcular IMC (backend y frontend) - ⚠️ Falta validación de `NaN` e `Infinity` después de conversiones

Recomendación: - Implementar validación de tipos antes de conversión - Verificar que los números sean finitos después de `float()` y `parseFloat()`

3.2.2. Riesgo MEDIO: Clickjacking (CWE-1021)

Descripción del riesgo: - La aplicación puede ser embebida en iframes maliciosos - Los usuarios pueden ser engañados para realizar acciones no deseadas - Posible robo de datos personales mediante superposición de elementos

Impacto en el negocio: - **Confidencialidad**: Los datos personales pueden ser comprometidos - **Integridad**: Los datos pueden ser modificados sin consentimiento del usuario - **Reputación**: Pérdida de confianza si se explota esta vulnerabilidad

Mitigación actual: - ❌ No hay headers de seguridad implementados

Recomendación: - Implementar `X-Frame-Options: DENY` - Agregar `Content-Security-Policy: frame-ancestors 'none'` - Incluir `X-Content-Type-Options: nosniff`

3.2.3. Riesgo MEDIO/ALTO: CORS Permisivo (CWE-942)

Descripción del riesgo: - Cualquier sitio web puede realizar peticiones a la API - Aumento significativo de la superficie de ataque en producción - Posibilidad de ataques CSRF desde dominios externos

Impacto en el negocio: - **Confidencialidad**: Datos pueden ser accedidos desde sitios maliciosos - **Integridad**: Datos pueden ser modificados mediante ataques CSRF - **Disponibilidad**: Posible denegación de servicio mediante peticiones masivas

Mitigación actual: - ⚠️ CORS configurado con `origins: "*"` (permite cualquier origen) - ✅ Actualmente solo accesible desde `localhost` (riesgo limitado)

Recomendación: - Restringir CORS a dominios específicos cuando se exponga el backend - Implementar validación de origen en producción - Considerar implementar tokens CSRF para operaciones sensibles

3.2.4. Riesgo BAJO: Manejo de Excepciones (CWE-703)

Descripción del riesgo: - Uso de `Exception` genérico puede ocultar errores inesperados - Dificultad para detectar y responder a vulnerabilidades - Información de error insuficiente para monitoreo

Impacto en el negocio: - **Mantenibilidad**: Dificultad para identificar y corregir problemas - **Seguridad**: Errores de seguridad pueden pasar desapercibidos

Mitigación actual: - ✅ Validación de nombres usa manejo estructurado de errores - ⚠️ Conversiones de `float()` aún usan `Exception` genérico

Recomendación: - Especificar excepciones concretas (`ValueError`, `TypeError`, `KeyError`) - Implementar logging detallado de errores

3.3. Resumen de Riesgos por Categoría

Riesgos Críticos: Ninguno identificado

Riesgos Altos: - CWE-942 (CORS) - Solo si se expone fuera de `localhost`

Riesgos Medios: - CWE-1287 (Type Confusion Backend) - CWE-843 (Type Confusion Frontend) - CWE-1021 (Clickjacking)

Riesgos Bajos: - CWE-703 (Manejo de Excepciones)

Riesgos Resueltos: - CWE-20 (Validación de Entrada) ✅

---

4. Nivel ASVS Seleccionado, con Justificación

4.1. OWASP Application Security Verification Standard (ASVS)

El **OWASP Application Security Verification Standard (ASVS)** es un estándar de seguridad para aplicaciones web que define tres niveles de verificación:

- **Nivel 1 (Básico)**: Para aplicaciones de bajo riesgo, con controles de seguridad básicos
- **Nivel 2 (Estándar)**: Para aplicaciones de riesgo estándar, con controles de seguridad estándar
- **Nivel 3 (Avanzado)**: Para aplicaciones de alto riesgo, con controles de seguridad avanzados

4.2. Nivel Seleccionado: NIVEL 2 (ESTÁNDAR)

4.3. Justificación de la Selección

4.3.1. Naturaleza de los Datos

La aplicación maneja **datos de salud personales** (peso, altura, fecha de nacimiento), aunque no sean datos médicos críticos ni información de identificación sensible. Según el Reglamento General de Protección de Datos (RGPD), los datos de salud están categorizados como **datos personales sensibles**, lo que requiere un nivel de protección superior al básico.

Justificación: Aunque los datos no son críticos, el hecho de ser datos de salud justifica un nivel de seguridad estándar (Nivel 2) en lugar de básico (Nivel 1).

4.3.2. Contexto de Uso

- **Monousuario**: La aplicación es monousuario, lo que reduce el riesgo de exposición de datos entre usuarios
- **Sin autenticación compleja**: No requiere autenticación robusta, lo que limita algunos vectores de ataque
- **Almacenamiento local**: Los datos se almacenan principalmente en localStorage, reduciendo el riesgo de exposición en el servidor

Justificación: El contexto monousuario y el almacenamiento local reducen algunos riesgos, pero no eliminan la necesidad de controles de seguridad estándar.

4.3.3. Vectores de Ataque Identificados

El análisis CWE-699 identificó **6 debilidades de seguridad**, de las cuales: - 1 está resuelta (CWE-20) - 4 requieren atención (CWE-1287, CWE-843, CWE-1021, CWE-942) - 1 está mejorada parcialmente (CWE-703)

Justificación: La presencia de múltiples debilidades de seguridad de severidad media justifica la implementación de controles de seguridad estándar (Nivel 2) en lugar de básicos (Nivel 1).

4.3.4. Requisitos de Cumplimiento

Aunque la aplicación no maneja datos médicos críticos, el manejo de datos de salud personales puede requerir cumplimiento con: - **RGPD**: Reglamento General de Protección de Datos - **Estándares de seguridad de aplicaciones web**: Buenas prácticas de la industria

Justificación: El cumplimiento con estándares de seguridad de la industria y protección de datos personales requiere al menos un nivel estándar de seguridad (Nivel 2).

4.3.5. Por qué NO Nivel 3 (Avanzado)

- No maneja datos médicos críticos
- No requiere autenticación multi-factor
- No tiene requisitos de alta disponibilidad
- No maneja transacciones financieras
- No es una aplicación crítica para la vida

Justificación: La aplicación no cumple con los criterios para requerir controles de seguridad avanzados (Nivel 3).

4.3.6. Por qué NO Nivel 1 (Básico)

- Maneja datos de salud personales (datos sensibles según RGPD)
- Identificadas múltiples debilidades de seguridad de severidad media
- Requiere protección contra vectores de ataque comunes (clickjacking, CORS, type confusion)

Justificación: La naturaleza de los datos y los vectores de ataque identificados requieren controles de seguridad más allá del nivel básico.

4.4. Requisitos ASVS Nivel 2 Aplicables

Basado en el análisis realizado, los siguientes requisitos ASVS Nivel 2 son relevantes para esta aplicación:

4.4.1. V1: Architecture, Design and Threat Modeling

- ✅ Arquitectura definida y documentada
- ✅ Análisis de amenazas realizado (CWE-699)

4.4.2. V2: Authentication

- ⚠️ No aplicable (aplicación monousuario sin autenticación compleja)

4.4.3. V3: Session Management

- ⚠️ No aplicable (sin sesiones)

4.4.4. V4: Access Control

- ⚠️ No aplicable (monousuario)

4.4.5. V5: Validation, Sanitization and Encoding

- ✅ Validación de entrada implementada (CWE-20 resuelto)
- ⚠️ Validación de tipos mejorable (CWE-1287, CWE-843 pendientes)

4.4.6. V6: Stored Cryptographically Sensitive Data

- ⚠️ No aplicable (datos almacenados localmente, no en servidor)

4.4.7. V7: Error Handling and Logging

- ⚠️ Manejo de errores mejorable (CWE-703 mejorado parcialmente)

4.4.8. V8: Data Protection

- ✅ Validaciones defensivas implementadas
- ⚠️ Headers de seguridad pendientes (CWE-1021)

4.4.9. V9: Communications

- ⚠️ CORS demasiado permisivo (CWE-942 pendiente)

4.4.10. V10: Malicious Code

- ✅ Validación y sanitización de entrada implementada

4.5. Conclusión del Nivel ASVS

Nivel seleccionado: **NIVEL 2 (ESTÁNDAR)**

Justificación resumida: 1. Maneja datos de salud personales (datos sensibles según RGPD) 2. Identificadas múltiples debilidades de seguridad de severidad media 3. Requiere controles de seguridad estándar para proteger datos personales 4. No requiere controles avanzados (Nivel 3) por no manejar datos críticos

Estado de cumplimiento: ⚠️ **PARCIAL** - Se requiere implementar mejoras para alcanzar el cumplimiento completo del Nivel 2.

---

5. Conclusión Final

5.1. Resumen Ejecutivo

La aplicación médica analizada es una herramienta web monousuario para el registro personal de peso e IMC. Aunque no maneja datos médicos críticos, sí procesa **datos de salud personales** que requieren protección adecuada según estándares de seguridad y protección de datos.

El análisis de seguridad basado en **CWE-699 (Software Development)** identificó **6 debilidades de seguridad**, de las cuales: - **1 está completamente resuelta** (CWE-20: Validación de entrada) - **4 requieren atención** (CWE-1287, CWE-843, CWE-1021, CWE-942) - **1 está mejorada parcialmente** (CWE-703: Manejo de excepciones)

5.2. Vectores de Ataque Principales

Los principales vectores de ataque identificados son:

1. **Type Confusion** (CWE-1287, CWE-843): Validación insuficiente de tipos numéricos que puede resultar en valores `NaN` o `Infinity` en cálculos
2. **Clickjacking** (CWE-1021): Falta de headers de seguridad que permite la inclusión de la aplicación en iframes maliciosos
3. **CORS Permisivo** (CWE-942): Configuración que permite peticiones desde cualquier origen, aumentando la superficie de ataque

5.3. Nivel de Riesgo General

El **riesgo general de la aplicación es MEDIO**, con los siguientes factores:

- **Riesgos Críticos**: Ninguno identificado
- **Riesgos Altos**: CWE-942 (solo si se expone fuera de `localhost`)
- **Riesgos Medios**: CWE-1287, CWE-843, CWE-1021
- **Riesgos Bajos**: CWE-703

5.4. Nivel ASVS Recomendado

Se recomienda implementar controles de seguridad según **OWASP ASVS Nivel 2 (Estándar)** debido a:

1. **Naturaleza de los datos**: Datos de salud personales (datos sensibles según RGPD)
2. **Vectores de ataque identificados**: Múltiples debilidades de severidad media
3. **Requisitos de cumplimiento**: Estándares de seguridad de la industria y protección de datos

Estado de cumplimiento: ⚠️ **PARCIAL** - Se requiere implementar mejoras para alcanzar el cumplimiento completo.

5.5. Recomendaciones Prioritarias

Para alcanzar el cumplimiento del **ASVS Nivel 2**, se recomienda implementar las siguientes mejoras en orden de prioridad:

Prioridad Alta:

1. **Implementar headers de seguridad** (CWE-1021)
2. `X-Frame-Options: DENY`
3. `Content-Security-Policy: frame-ancestors 'none'`

4. `X-Content-Type-Options: nosniff`

5. **Validar NaN e Infinity** en frontend y backend (CWE-1287, CWE-843)

6. Validar tipos antes de conversión
7. Verificar que los números sean finitos después de `float()` y `parseFloat()`

Prioridad Media:

1. **Restringir CORS en producción** (CWE-942)
2. Limitar orígenes permitidos cuando el backend se exponga fuera de `localhost`

3. Implementar validación de origen

4. **Mejorar manejo de excepciones** (CWE-703)

5. Especificar excepciones concretas en lugar de `Exception` genérico
6. Implementar logging detallado de errores

5.6. Evaluación Final

La aplicación implementa **buenas prácticas de seguridad** en validación de entrada y manejo defensivo de datos. Sin embargo, requiere **mejoras en validación de tipos, headers de seguridad y configuración CORS** para alcanzar el cumplimiento completo del **ASVS Nivel 2**.

Estado general: ⚠️ **BUENO CON ÁREAS DE MEJORA IDENTIFICADAS**

Las debilidades identificadas son de **severidad media o baja** y pueden corregirse **sin cambios arquitectónicos significativos**, lo que facilita su implementación y mejora la postura de seguridad de la aplicación.

Fecha de informe: 2025-01-27
Analista: Análisis automatizado basado en CWE-699
Referencias: - CWE-699: Software Development - OWASP ASVS - OWASP Top 10