ESCUELA POLITÉCNICA NACIONAL

ISWD853 Desarrollo de software seguro

ESTUDIANTES:	Mateo Pilco, Joel Piuri, José Terán, Juan Rengifo, Iván Simbaña
FECHA:	03-08-2025
TEMA:	Informe de análisis estático de seguridad.
PROYECTO:	CORE BANCARIO API.

Examen Segundo Bimestre Parte 2
GRUPO DevSec

Repositorio: https://github.com/SebasPM15/examen-devsec-2025a-GR2

Tareas para realizar:
- Específicos: TCE-04
- Funcionalidades y Seguridad: TCG-01 Y TCG-02
- Reportes: TRG-01

Tarea TRG-01 REPORTES

Descripción:
TRG-01 – Realizar un análisis estático del código, elaborar un informe que contenga la siguiente
estructura mínima. El informe debe ser 100% de su autoría, en caso de no cumplirse esto se penalizará con un 45% de descuento de la nota: (200 p).
- Nombres de los estudiantes que efectivamente realizan esta parte.
- Resumen ejecutivo (Gráfica que muestra la cantidad de vulnerabilidades encontradas)
- Hallazgos
    - Código de identificación del hallazgo
    - Descripción de la vulnerabilidad
    - Severidad (Calculada usando probabilidad por impacto o CVSS)
    - Recomendación / Mecanismo de mitigación

DESARROLLO

Para el desarrollo del análisis estático se hizo uso de 5 herramientas dedicadas a este apartado las cuales son:

- SonarQube cloud
- Bandit
- Safety
- Semgrep
- Análisis Manual

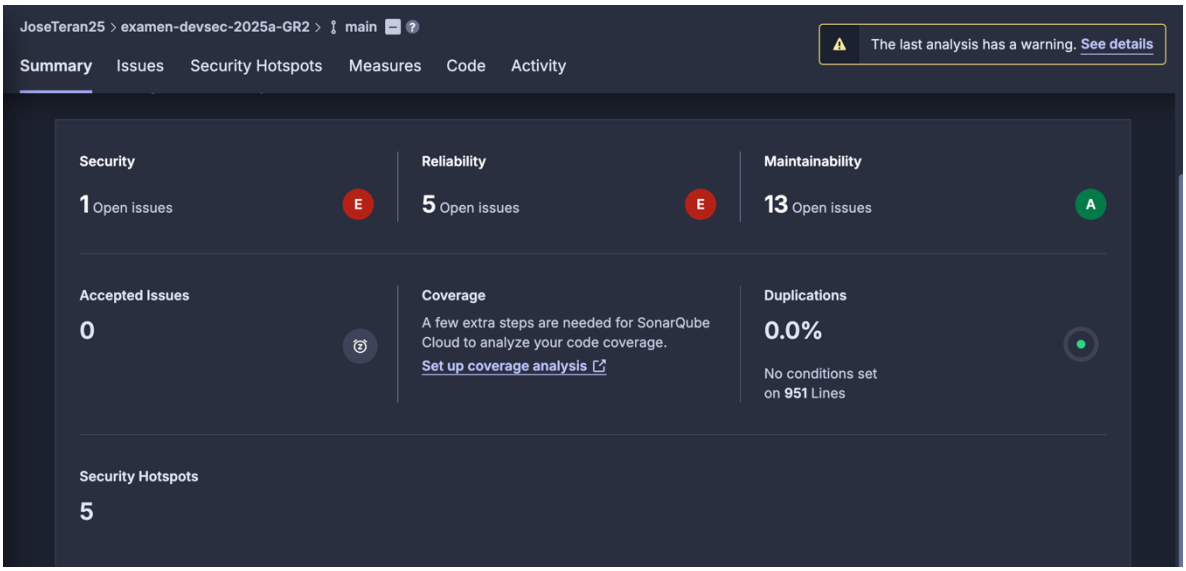
RESUMEN

Con los resultados del análisis estático se realizó una cobertura completa del código fuente hasta las dependencias del proyecto.

1. Alcance

    - 951 líneas de código analizadas por SonarCloud
    - 6 módulos principales dentro del directorio /app
    - Análisis de dokcer compose, Dockerfile, requirements.txt
    - 7 librerías de Python verificadas

2. Gráficas
    a. SonarQube Cloud



b. Banidt

Metrics:
Total lines of code: 737
Total lines skipped (#nosec): 0

flask_debug_true: A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
Test ID: B201
Severity: HIGH
Confidence: MEDIUM
CWE: CWE-94
File: app/main.py
Line number: 516
More info: https://bandit.readthedocs.io/en/1.8.6/plugins/b201_flask_debug_true.html

```
515     if __name__ == "__main__":
516         app.run(host="0.0.0.0", port=8000, debug=True)
517
```

hardcoded_bind_all_interfaces: Possible binding to all interfaces.
Test ID: B104
Severity: MEDIUM
Confidence: MEDIUM
CWE: CWE-605
File: app/main.py
Line number: 516
More info: https://bandit.readthedocs.io/en/1.8.6/plugins/b104_hardcoded_bind_all_interfaces.html

```
515     if __name__ == "__main__":
516         app.run(host="0.0.0.0", port=8000, debug=True)
517
```

blacklist: Standard pseudo-random generators are not suitable for security/cryptographic purposes.
Test ID: B311
Severity: LOW
Confidence: HIGH
CWE: CWE-330
File: app/utils.py
Line number: 41
More info: https://bandit.readthedocs.io/en/1.8.6/blacklists/blacklist_calls.html#b311-random

```
40      def generate_otp(length: int = 6) -> str:
41          return ''.join(random.choices(string.digits, k=length))
42
```

c. Safety

Safety Check Report

Scan Summary

Packages Found (details ↓)	Vulnerabilities Reported (details ↓)
68	4

Meta-data

Time: 2025-08-03 12:22:24
Safety version: 3.6.0
Use an API key
Configuration file: None

Scan target: environment
Scan paths:
['/Users/josedavidteranramos/Documents/Jose/Universidad/Proyecto software Seguro/Sin título/examen-devsec-2025a-GR2',
'/Users/josedavidteranramos/Library/Python/3.9/lib/python/site-packages/setuptools/_vendor',
'/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/lib/python39.zip',
'/Library/Developer/CommandLineTools/Library/Frameworks/Python3.fr

d. Semgrep

Reporte Semgrep

python.flask.security.audit.app-run-param-config.avoid_app_run_with_bad_host

Archivo: app/main.py

Línea: 516

Mensaje: Running flask app with host 0.0.0.0 could expose the server publicly.

---

python.flask.security.audit.debug-enabled.debug-enabled

Archivo: app/main.py

Línea: 516

Mensaje: Detected Flask app with debug=True. Do not deploy to production with this flag enabled as it will leak sensitive information. Instead, consider using Flask configuration variables or setting 'debug' using system environment variables.

---

e. Análisis Manual

3. Dashboard general de vulnerabilidades

Herramienta	Crítica	Alta	Media	Baja
SonarCloud	1	1	1	1
Bandit	-	1	1	1
Safety	2	2	-	-
Semgrep	-	2	-	-
Manual	1	-	1	1
Total	4	6	2	1

HALLAZGOS DETALLADOS

1) VULNERABILIDADES CRÍTICAS

a) VUL-001: Credenciales Hardcodeadas en Configuración

Severidad: CRÍTICA
Ubicación: docker-compose.yml:29-30
Herramienta: Análisis Manual + SonarCloud
CWE: CWE-798 (Use of Hard-coded Credentials)

Descripción: Claves críticas de encriptación (FERNET_KEY) y autenticación (JWT_SECRET) están hardcodeadas directamente en el archivo docker-compose.yml, expuestas en el repositorio público. SonarCloud confirma específicamente "Make sure this Fernet key gets revoked, changed, and removed from the code".

Impacto:

- Compromiso total de la seguridad de encriptación de datos
- Tokens JWT predecibles y falsificables
- Acceso no autorizado a funciones administrativas
- Violación grave de estándares de seguridad bancaria

Recomendación: Se debe migrar inmediatamente todas las credenciales a variables de entorno, implementar Docker Secrets o gestores como HashiCorp Vault, rotar todas las claves expuestas y establecer políticas de gestión de secretos con archivos .env no versionados.

b) VUL-002: Command Injection en pip

Severidad: CRÍTICA
Ubicación: pip==21.2.4
Herramienta: Safety
CVE: CVE-2023-5752

Descripción: La versión de pip utilizada (21.2.4) es vulnerable a inyección de comandos al instalar paquetes desde URLs Mercurial VCS, permitiendo ejecución de configuraciones arbitrarias.

Impacto:

- Ejecución remota de comandos durante instalación
- Compromiso del entorno de desarrollo/producción
- Modificación de repositorios de código

Recomendación: Es crítico actualizar pip a versión 23.3 o superior inmediatamente y realizar auditoría de todas las instalaciones previas desde URLs VCS para verificar la integridad del sistema.

c) VUL-003: Ejecución de Código en Archivos Wheel

Severidad: CRÍTICA
Ubicación: pip==21.2.4
Herramienta: Safety
ID: 75180

Descripción: Pip permite ejecución de código no autorizado durante instalación de archivos wheel maliciosamente crafteados.

Impacto:

- Ejecución arbitraria de código durante deploy
- Compromiso silencioso del sistema
- Instalación de backdoors persistentes

Recomendación: Se requiere actualizar pip a versión 25.0 o superior e implementar mecanismos de verificación de integridad y firma digital para todos los paquetes antes de su instalación.

d) VUL-004: Exposición de Claves en Logs

Severidad: CRÍTICA
Ubicación: app/utils.py:11-12
Herramienta: Análisis Manual
CWE: CWE-532 (Information Exposure Through Log Files)

Descripción: El sistema imprime la FERNET_KEY generada dinámicamente en consola cuando no encuentra la variable de entorno, exponiendo la clave de encriptación en logs del sistema.

Impacto:

- Filtración de clave de encriptación en logs del sistema
- Posible descifrado de datos sensibles almacenados
- Comprometimiento de la confidencialidad de datos financieros

Recomendación: Es necesario eliminar completamente todos los prints de información sensible, implementar un sistema de logging seguro que no exponga credenciales y configurar rotación automática con cifrado de logs.

2) VULNERABILIDADES ALTAS

a) VUL-005: Flask Debug Mode Habilitado

Severidad: ALTA
Ubicación: app/main.py:516
Herramienta: Bandit + Semgrep + Manual
CWE: CWE-94 (Code Injection)

Descripción: La aplicación Flask se ejecuta con debug=True, exponiendo el debugger interactivo Werkzeug que permite ejecución de código Python arbitrario.

Impacto:

- Ejecución remota de código via debugger web
- Exposición de stack traces con información sensible
- Acceso a variables de entorno y configuración

Recomendación: Se debe deshabilitar inmediatamente el modo debug en producción, implementar configuraciones separadas por entornos y desarrollar un sistema de manejo de errores personalizado que no exponga información interna.

b) VUL-006: Binding a Todas las Interfaces

Severidad: ALTA
Ubicación: app/main.py:516

Herramienta: Bandit + Semgrep + Manual
CWE: CWE-605 (Multiple Binds to the Same Port)

Descripción: La aplicación se ejecuta con host="0.0.0.0", exponiendo el servicio a todas las interfaces de red, incluyendo interfaces públicas.

Impacto:

- Exposición pública no intencionada del servicio
- Acceso desde cualquier interfaz de red
- Ampliación de superficie de ataque

Recomendación: Cambiar la configuración a host="127.0.0.1" para desarrollo local, implementar un proxy reverso como nginx para producción y configurar reglas de firewall restrictivas con segmentación de red apropiada.

c) VUL-007: Bug de Confiabilidad en app/main.py

Severidad: ALTA
Ubicación: app/main.py
Herramienta: SonarCloud
Tipo: Reliability Bug (Rating E)

Descripción: SonarCloud identifica un bug específico en app/main.py que causa Reliability Rating E, relacionado con manejo de errores y operaciones de base de datos que pueden fallar en tiempo de ejecución.

Impacto:

- Fallos potenciales en operaciones bancarias críticas
- Errores no manejados en transacciones financieras
- Inconsistencias en el estado de la aplicación

Recomendación: Revisar y corregir el bug específico identificado por SonarCloud, implementar manejo robusto de excepciones en todas las operaciones críticas y añadir validaciones exhaustivas antes de ejecutar transacciones de base de datos.

d) VUL-008: Vulnerabilidad DoS en wheel

Severidad: ALTA
Ubicación: wheel==0.37.0
Herramienta: Safety
CVE: CVE-2022-40898

Descripción: Vulnerabilidad de Denial of Service en wheel CLI via input controlado por atacante.

Impacto:

- Denial of Service del sistema durante instalación de paquetes
- Interrupción de servicios críticos

Recomendación: Actualizar inmediatamente la librería wheel a versión 0.38.1 o superior para resolver la vulnerabilidad de denegación de servicio identificada.

e) VUL-009: Vulnerabilidad DoS en future

Severidad: ALTA
Ubicación: future==0.18.2
Herramienta: Safety
CVE: CVE-2022-40899

Descripción: DoS via Set-Cookie header malicioso en servidores web maliciosos.

Impacto:

- Denial of Service via headers HTTP maliciosos
- Interrupción de servicios web

Recomendación: Actualizar la librería future a una versión superior a 0.18.2 y considerar implementar validación y filtrado de headers HTTP para prevenir ataques de denegación de servicio.

3) VULNERABILIDADES MEDIAS

a) VUL-010: Security Hotspots SonarCloud

Severidad: MEDIA
Ubicación: 5 áreas específicas
Herramienta: SonarCloud
Tipo: Security Hotspots

Descripción: SonarCloud identifica 5 Security Hotspots que requieren revisión manual para determinar si constituyen vulnerabilidades reales.

Impacto:

- Posibles vulnerabilidades según implementación específica
- Riesgo contextual variable

Recomendación: Realizar revisión manual exhaustiva de cada hotspot identificado en el dashboard de SonarCloud, documentar las decisiones de aceptación o mitigación, e implementar controles compensatorios donde sea técnicamente necesario.

b) VUL-011: Generador Pseudoaleatorio Inseguro para OTPs

Severidad: MEDIA
Ubicación: app/utils.py:41
Herramienta: Bandit
CWE: CWE-330 (Use of Insufficiently Random Values)

Descripción: El sistema usa random.choices() para generar códigos OTP, que no es criptográficamente seguro y puede ser predecible.

Impacto:

- OTPs predecibles para autenticación bancaria
- Bypass de autenticación de dos factores
- Compromiso de cuentas de usuario

Recomendación: Reemplazar el generador actual con secrets.choice() o os.urandom() para asegurar generación criptográficamente segura, implementar algoritmos estándar TOTP/HOTP y auditar todos los demás usos de generación aleatoria en el sistema.

4) VULNERABILIDADES BAJA

a) VUL-012: Code Smells SonarCloud

Severidad: BAJA
Ubicación: 13 issues múltiples
Herramienta: SonarCloud
Tipo: Maintainability (Rating A)

Descripción: A pesar de 13 code smells, el proyecto mantiene un excelente rating A de mantenibilidad, indicando que son issues menores.

Impacto:

- Dificultad menor de mantenimiento del código
- Impacto mínimo en calidad general

Recomendación: Resolver gradualmente estos issues durante los ciclos normales de desarrollo, priorizando aquellos que afecten la legibilidad del código y manteniendo el estándar alto actual mediante procesos de code review estructurados.

CONCLUSIONES

El análisis estático del Core Bancario API, realizado con cinco herramientas, identificó 12 vulnerabilidades (4 críticas). Aunque el sistema tiene buena arquitectura y mantenibilidad (Rating A), presenta fallas graves en seguridad, como credenciales hardcodeadas y dependencias vulnerables, con calificaciones E en seguridad y confiabilidad.

Actualmente no es apto para producción, pero con un plan de remediación de 2 a 4 semanas, puede cumplir estándares bancarios gracias a su base técnica sólida.